

TEC-0122

Prototyping the DARPA Image Understanding Environment (IUE) and Tools to Facilitate Its Use

Daryl T. Lawton Ted Rathkopf
Warren Gardner Shumei Lin

Georgia Institute of Technology
G.V.U. Center
Atlanta, GA 30332-0280

May 1999

19990611 079

Approved for public release; distribution is unlimited.

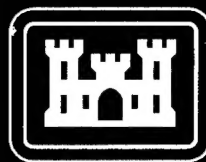
Prepared for:

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

Monitored by:

U.S. Army Corps of Engineers
Topographic Engineering Center
7701 Telegraph Road
Alexandria, Virginia 22315-3864

DTIC QUALITY INSPECTED 4

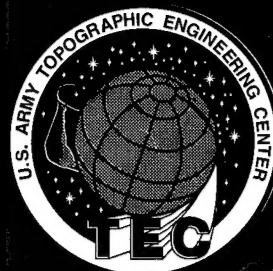


US Army Corps
of Engineers
Topographic
Engineering Center

T

E

C



**Destroy this report when no longer needed.
Do not return it to the originator.**

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE May 1999	3. REPORT TYPE AND DATES COVERED Final Technical September 1993 - August 1995	
4. TITLE AND SUBTITLE Prototyping the DARPA Image Understanding Environment (IUE) and Tools to Facilitate Its Use			5. FUNDING NUMBERS DACA76-92-C-0016	
6. AUTHOR(S) Daryl T. Lawton, Warren Gardner, Ted Rathkopf, Shumei Lin				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Georgia Institute of Technology G.V.U. Center Atlanta, GA 30332-0280			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive, Arlington, VA 22203-1714 U.S. Army Topographic Engineering Center 7701 Telegraph Road, Alexandria, VA 22315-3864			10. SPONSORING / MONITORING AGENCY REPORT NUMBER TEC-0122	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report is the final report of the research on Prototyping the DARPA Image Understanding Environment (IUE) and Tools to Facilitate Its Use. The major objectives of this project were to support the design and development of the IUE, to prototype the IUE user interface and data exploration tools, and to develop tools for documentation, tutorials, and publication.				
14. SUBJECT TERMS Technology transfer, annotation, interface design, prototyping			15. NUMBER OF PAGES 131	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNLIMITED	

Table of Contents

	TITLE	PAGE
	Preface	ix
1	Overview.....	1
1.1	Project Overview	1
1.1.1	Second and Third Year Accomplishments	1
1.2	IUE Interface Design and Prototyping.....	2
1.3	World Wide Web Tools	2
1.4	Knowledge Weasel.....	3
1.5	Shape and Motion from Linear Features.....	3
1.6	Interactive Model-Based Vehicle Tracking	3
2	IUE Interface Objects Prototype	5
2.1	Image Display.....	5
2.1.1	Program Fragment	8
2.2	Interactive Threshold Selection	9
2.2.1	Program Fragment	10
2.3	ORB with Different Mapping Functions.....	10
2.3.1	Program Fragment	13
2.4	Zooming with the ORB.....	14
2.4.1	Program Fragment	14
2.5	Edges Over the Image.....	16
2.5.1	Program Fragment	17
2.6	Laplacian	19
2.6.1	Program Fragment	19
2.7	Links	20
2.7.1	Program Fragment	22
2.8	Snake	22
2.8.1	Program Fragment	24
2.9	Graph Browser	27
3	World Wide Web Tools.....	33
3.1	WWW Image Database	33
3.2	Proceedings of the 1994 DARPA Image Understanding Workshop.....	37
3.3	Snakes Tutorial	42
4	Knowledge Weasel.....	51
4.1	Components.....	51
4.2	Example of Use.....	52
4.3	The Database Browser.....	65
5	Gravity	69
5.1	Recovery of Gravity	69
5.2	Notation	70
5.3	Structure and Motion Estimation	71
5.3.1	Horizontal Lines.....	71

5.3.2	Line Orientation and Camera Rotation.....	72
5.3.3	Line Position and Camera Translation.....	73
5.4	Error Analysis	74
5.4.1	Errors from Reduced Image Resolution.....	74
5.4.2	Gravity Measurement Errors.....	77
5.4.3	Recovery of Structure	78
5.5	Results	80
5.5.1	Line Extraction and Matching.....	84
5.5.2	Scene Reconstruction.....	86
6	Model-Based Vehicle Tracking.....	91
6.1	System Architecture.....	91
6.2	Object Models.....	93
6.2.1	Road Model.....	93
6.2.2	Gravity Model	93
6.2.3	Vehicle Model	95
6.3	Difference Tracker.....	96
6.4	Local Translation Tracker	98
6.4.1	Feature Extraction.....	100
6.4.2	Feature Extraction from a Model	101
6.4.3	Locally Planar Motion	105
6.4.4	Temporal Filtering	106
6.4.5	Results.....	108

List of Figures

FIGURE	PAGE
2.1 Original image in DisplayObject	5
2.2 Darkened image.....	6
2.3 Lightened image	6
2.4 Interpolated image.....	7
2.5 Colored image	7
2.6 Interactive threshold selection.....	9
2.7 Thresholded image	9
2.8 Navigation control panel.....	10
2.9 ORB for original image.....	11
2.10 ORB for darkened image	11
2.11 ORB for lightened image	12
2.12 ORB for interpolated image	12
2.13 ORB for colored image	12
2.14 ORB zooming.....	14
2.15 Zooming with the Photo Widget.....	16
2.16 Application-based zooming	17
2.17 Laplacian image.....	19
2.18 Linked DisplayObject window.....	21
2.19 Linked ORB canvas.....	21
2.20 Snake DisplayObject window.....	23
2.21 Snake ORB canvas.....	23
2.22 Initialized graph browser	27
2.23 Creating a graph	28
2.24 Searching a node	28
2.25 Links from node 0.....	29
2.26 Traversing a path of the graph.....	29
2.27 Saving a path of the graph.....	30
2.28 Inspecting node attributes	31
2.29 Inspecting graph path	32
2.30 Showing visited nodes.....	32
3.1 Image database browser	33
3.2 Image header	34
3.3 Image submission page (top)	35
3.4 Image submission page (bottom).....	36
3.5 Proceedings of the 1994 DARPA IUW home page	37
3.6 Table of contents	38
3.7 Author index	38
3.8 Organization index	39
3.9 Organization home page	39
3.10 Paper abstract	40
3.11 Search result	41
3.12 Snakes home page.....	42
3.13 Processing examples.....	43
3.14 Initial point positions for the snake.....	44

3.15	Initial snake position	45
3.16	Final snake position.....	45
3.17	Snake position at different iterations	46
3.18	Code fragment from the tutorial	47
3.19	Explanation associated with the variable nextNode	47
3.20	Snakes tutorial home page.....	48
3.21	Explanation of neighborhood	49
4.1	Main control panel.....	52
4.2	File menu	53
4.3	Commit File window	53
4.4	Canvas file type.....	54
4.5	Bitmap display	55
4.6	Saving a canvas file	55
4.7	File selector	56
4.8	Annotate menu	56
4.9	Annotation attribute menu	57
4.10	Annotation file window.....	58
4.11	Annotation attributes for triangle eBf	59
4.12	Overlay for triangle eBf.....	59
4.13	Graphical viewer window for triangle eBf	60
4.14	Annotation tree viewer	61
4.15	Selecting a "layer"	62
4.16	Display of multiple overlays	63
4.17	Tree view of multiple overlays.....	64
4.18	Database Browser	65
4.19	Attribute Selector	66
4.20	Query Keypad	66
4.21	Mapper	66
4.22	Example query with two attributes selected for viewing.....	67
4.23	Selecting font and colors for the mapping.....	68
4.24	Example query mapping.....	68
5.1	Gravity coordinate system.....	70
5.2	Frames 1, 25, and 50 from a 40 image sequence.....	74
5.3	A single image frame produced using different image sizes.....	75
5.4	Angle of rotation computed from different image sizes.....	75
5.5	Angle of rotation computed from a noisy gravitational axis.....	77
5.6	Top view extracted from 512x512 images and no gravity error	78
5.7	Top view extracted from 128x128 images and no gravity error	79
5.8	Top view extracted from 512x512 images and 4 degrees gravity errors.....	79
5.9	Top view extracted from 128x128 images and 4 degrees gravity error.....	80
5.10	1st image from a 50 frame sequence	81
5.11	25th image from a 50 frame sequence	82
5.12	50th image from a 50 frame sequence	83
5.13	Image 1 after non-maxima suppression.....	84
5.14	Hough space corresponding to image 1	84
5.15	Lines extracted from image 1	85
5.16	Lines extracted from image 25.....	85
5.17	Lines extracted from image 50.....	86
5.18	Derived angle of rotation	86
5.19	Perspective view #1 of the reconstructed scene.....	87
5.20	Perspective view #2 of the reconstructed scene.....	87
5.21	Perspective view of the books	88

6.1	Model-based system architecture	92
6.2	2-D road model	93
6.3	Gravity model projected onto an image	94
6.4	User interface for the gravity model	95
6.5	Perspective view of the vehicle model	95
6.6	Image areas lying near the road model are smoothed	96
6.7	Areas of motion extracted by region growing	97
6.8	Center of the areas of motion	97
6.9	Two dimensional motion trajectory	98
6.10	Vehicle tracking system architecture	99
6.11	Image of a truck	100
6.12	Zero crossings and features extracted from the truck image	101
6.13	Features grouped using a vehicle model	102
6.14	User-instantiated vehicle model	103
6.15	Sub-images extracted using the vehicle model	104
6.16	Convolved sub-images	104
6.17	Adjusted vehicle model position	105
6.18	User-instantiated gravity model	108
6.19	User-instantiated gravity model overlayed on image frame 1	109
6.20	User-instantiated vehicle model	110
6.21	Adjusted vehicle model for image frame 1	111
6.22	Extracted and grouped features for image frame 1	112
6.23	Local translation vectors for the three feature groups	113
6.24	Vehicle model and features for image frame 10	114
6.25	Vehicle model and features for image frame 20	115
6.26	Vehicle model and features for image frame 30	116
6.27	Vehicle model and features for image frame 40	117
6.28	Vehicle trajectory for the 45 frame sequence	118
6.29	Height of the vehicle above the gravity plane	119
6.30	Vehicle orientation	119
6.31	Linear velocity of the vehicle	120
6.32	Angular velocity of the vehicle	120

List of Tables

	TABLE	PAGE
5.1	Reconstructed line measurements	88
5.2	Reconstructed plane measurements	89

Preface

This report describes research done during the second and final year of the project "Prototyping the DARPA Image Understanding Environment and Tools to Facilitate Its Use." This research is supported by the Defense Advanced Research Projects Agency (DARPA) of the Department of Defense and is monitored by the U. S. Army Topographic Engineering Center under Contract No. DACA76-92-C-0016. The DARPA program Manager is Mr. Oscar Firschien; and the TEC Contractor Officers' Representative is Ms. Laretta Williams.

Prototyping the DARPA Image Understanding Environment (IUE) and Tools to Facilitate Its Use

Chapter 1

1.1 Project Overview

The DARPA Image Understanding Environment (IUE) is a common software environment for image understanding research that will support the transfer of technology and research from the DARPA Image Understanding community. The major objectives of this project are to support the design and development of the IUE; prototype the IUE user interface and data exploration tools; and develop tools for documentation, tutorials, and publication that will facilitate the impact and the widespread adoption of the DARPA IUE.

The primary concerns of the work on this project are:

- The overall design of the IUE through activities of the IUE Design Committee ("The Gang of Ten")
- Prototyping the IUE user interface and data exploration tools
- Developing tools for documentation, tutorials, and publication that will facilitate the impact and the widespread adoption of the DARPA IUE
- Basic Vision research by students supported by the contract.

In the first three sections of this report, the initial prototyping of the user interface of the DARPA IUE, and the tools being developed for documentation, tutorials and publication that will facilitate the use and adoption of the IUE, are presented. The following chapters contain descriptions of work on motion processing using the direction of gravity and tracking algorithms in outdoor environments that use object models.

1.1.1 Second and Third Year Accomplishments

- Completed interface prototyping
- Completed development of prototype hypermedia annotation tools for on-line IUE documentation and tutorials
- Completed evaluation of publicly available hyper-text systems for on-line IUE documentation and tutorials
- Developed prototype tutorials using hyper-media annotation tools or publicly available hyper-text systems

- Produced an online, Internet-accessible version of DARPA IU Workshop Proceedings, as well as a CD-ROM version
- Produced an online, Internet accessible Image Database
- Began to organize materials for on-line tutorials
- Continued research in motion processing and interactive model-based vision systems.

Following is an overview of the different sections of this report.

1.2 IUE Interface Design and Prototyping

The user interface of the IUE is intended to provide flexible, simple, and powerful tools for exploring data, algorithms, and systems. The basic functional components of the IUE interface that have been described are:

- **Displays:** These deal with mapping spatial objects and images (or sets of spatial objects and images) onto 2-D display windows. There are several types of displays for displaying images and image-registered features, plotting functional relations between attributes and components of spatial objects, and displaying surfaces.
- **Browsers:** These deal with presenting textual and symbolic information about objects. There are different types of browsers for such operations as inspecting the values in a spatial object, performing interactive queries with respect to databases and sets of objects, and inspecting relational graphs and networks.
- **Interface Context Descriptors:** These are for describing the state of the interface and interface objects. Examples are such things as the current color-look-up table for a given display, current display window or browser, and links between interface objects that describe related views.
- **Command Language and Command Buffer:** Users can control their interaction with objects using an interactive command language. This also provides a complete description of the functionality of the user interface.
- **Simplified, programmable access to GUI objects:** This is intended to provide programmer access to several of the objects commonly found in Graphical User Interface (GUI) Construction Kits, such as knobs, sliders, text buffers, and menus. These can then be used in applications and to extend the interface

The display and browser objects were implemented using Tk/Tcl in C++, and processing examples are presented in this report. Tk/Tcl is a very rich machine-independent tool-kit for developing interactive user interfaces using an object-oriented language, that provides an interpretive command language and direct access to a wide variety of GUI objects.

1.3 World Wide Web Tools

A major part of this project is to develop authoring tools for producing documentation, demonstrations, and tutorials. These will be used for on-line documentation of the IUE and to support publication of research. Currently, the yearly proceedings from the DARPA IU Workshop

are a major source of technical output from the IU community. The IUE and the modules developed in this project will extend this significantly. People will have access to a much wider range of information than is currently contained in published proceedings. This will include such things as code, data, slides, viewgraphs, and tutorials developed by the authors themselves available on-line and through CD-ROM.

The HyperText Mark-Up Language (HTML) and the World Wide Web (WWW) are used to prototype several of these items. This includes: 1) an online, Internet-accessible image database; 2) an online, Internet-accessible version of the 1994 DARPA Image Understanding Workshop; and 3) a prototype tutorial describing snakes, which is online and Internet accessible. In addition, a CD-ROM version of the IU Workshop proceedings was produced and distributed at the most recent IU Workshop.

1.4 Knowledge Weasel

"Knowledge Weasel" (KW), a presentation and authoring system designed to support annotation using several different types of media, continued to be developed. Current implementation is in Tcl/Tk, with significant additions for browsing annotations.

1.5 Shape and Motion from Linear Features

Measurements of the direction of gravity can be fused with visual input to provide more accurate motion estimates, and to draw conclusions about environmental structure. An algorithm for computing structure and motion from line correspondences was developed, and is presented in Chapter 5. Since matching lines do not require the determination of exact 2D displacements, line features are preferred over point features. In order to constrain the possible motion configurations, it is assumed that the 3-D direction of gravity relative to each image frame is known. The algorithm presented is linear and can incorporate an arbitrary number of images, resulting in robust estimation of the parameters of motion.

Interestingly, psychophysical studies have shown that the perception of balance is important to the human visual system. These experiments suggest that gravity is used by the visual system for determining environmental orientation.

1.6 Interactive Model-Based Vehicle Tracking

While most work in motion processing has involved very minimal assumptions about subjects such as rigidity, a very important area for future work is motion processing, which incorporates object models. Investigation continues in the restricted domain of tracking vehicles from a stationary camera in outdoor road scenes. The key idea is that motion is a critical source of information for instantiating object models and that motion processing is in turn simplified by the constraints supplied by object models.

Processing begins with a human forming a rough interpretation of a scene by interactively manipulating models of objects such as terrain surface patches, roads, gravity, and vehicles. This initial, human-directed interpretation consists of incompletely specified 2-D drawings of expected image features and associated 3-D object models, which also are initially incompletely specified. Once an interpretation is in place, tracking algorithms then autonomously refine and extend the interpretation. For example, a human will indicate that a particular area is a road as a 2-D drawing. The system will then track movement along the road and fit a constraint-based description of a vehicle to this movement. As vehicles are tracked, the 3-D shape of the road can be recovered.

The system can determine that a vehicle has just gone off the road (or that it is behaving inconsistently with respect to the model of a vehicle) and report back to a human about unusual occurrences or behavior that cannot be accounted for.

Object models are related by constraints specifying necessary geometrical properties and relationships between objects. The use of constraints allows for flexible object instantiation. A user can indicate a vehicle, which directs perceptual processing routines to determine the corresponding local surface orientation and roads, or the user can instantiate a road segment to direct the extraction and tracking of vehicles.

Chapter 2

IUE Interface Objects Prototype

2.1 Image Display

The DisplayObject class in the IUE provides the basis for displaying images and applying composite linear mapping functions. This example presents the DisplayObject class and illustrates the effects of applying some mapping functions.

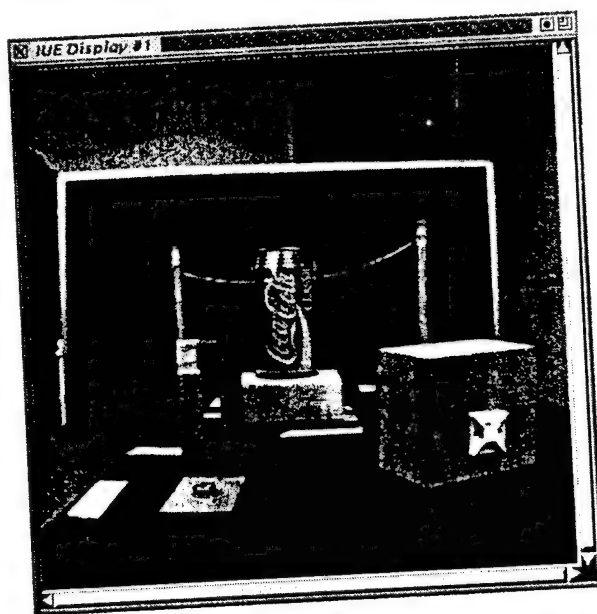


Figure 2.1. Original image in DisplayObject.

Figure 2.1 shows the image that will be used throughout this tutorial. A simple default mapping function is used here. The default mapping function maps image pixel values [min, max] to intensity values [0, 255]. The following four figures show the same image with different applied mapping functions. These figures illustrate the concept of mapping functions, and other mapping methods that can easily be implemented.

Figure 2.2 shows a darkened image where the image pixel values $[0.0, 255.0]$ are mapped to intensity values $[0, 150]$.



Figure 2.2. Darkened image.

Figure 2.3 shows a lightened image where the image pixel values $[0.0, 255.0]$ are mapped to intensity values $[100, 255]$.

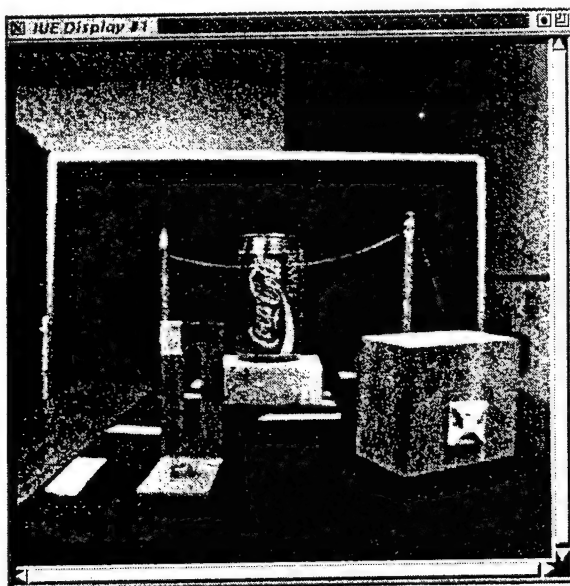


Figure 2.3. Lightened image.

Figure 2.4 shows an interpolated image where the image pixel values $[100.0, 200.0]$ are mapped to intensity values $[0, 255]$.

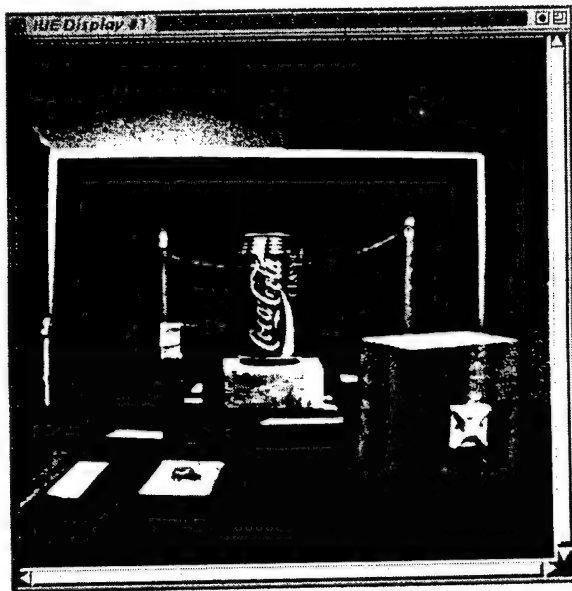


Figure 2.4. Interpolated image.

Figure 2.5 shows a colored image where image pixel values less than 128 are mapped to a blue color, and image pixel values greater than, or equal to, 128 are mapped to a red color.



Figure 2.5. Colored image.

2.1.1 Program Fragment

```
void
example1()
{
    DisplayObject *obj1;
    ImageDouble *im;
    char name[50];

    /*
     * Display image with default linear mapping parameters.
     * Map the range between minval and maxval to 0 and 255.
     * The clut will linearly interpolate colors from
     * (0, 0, 0) to (255, 255, 255) for the 255 entries.
     */
    obj1 = distrib->create_window(256, 256);
    cout << "please input PPM file name" << endl;
    cin >> name;
    im = distrib->load_image(name);
    obj1->set_clut(0, 256, 0, 0, 0, 255, 255, 255);
    obj1->set_intensity(im, 1.0, 1.0);
    obj1->display();
    distrib->EventLoop();

    /*
     * Apply different mapping parameters on the
     * current buffer to make it darker
     */
    obj1->set_mapping(0.0, 255.0, 0, 150);
    obj1->display();
    distrib->EventLoop();

    /*
     * Apply different mapping parameters on the
     * current buffer to make it lighter
     */
    obj1->set_mapping(0.0, 255.0, 100, 255);
    obj1->display();
    distrib->EventLoop();

    /*
     * Apply different mapping parameters to the current buffer to
     * interpolate more between 100.0 and 200.0.
     */
    obj1->set_mapping(100.0, 200.0, 0, 255);
    obj1->display();
    distrib->EventLoop();

    /*
     * Change clut mapping
     */
    obj1->default_mapping();
    obj1->set_clut(0, 128, 0, 0, 255, 0, 0, 0);
}
```

```
obj1->set_clut(128, 128, 0, 0, 0, 255, 0, 0);
obj1->display();
distrib->EventLoop();
```

2.2 Interactive Threshold Selection

It is sometimes desirable to apply thresholding to an image. The IUE provides interactive capabilities for selecting threshold values. The user clicks on any point in the image, then the intensity value of that point can be used in later thresholding procedures.

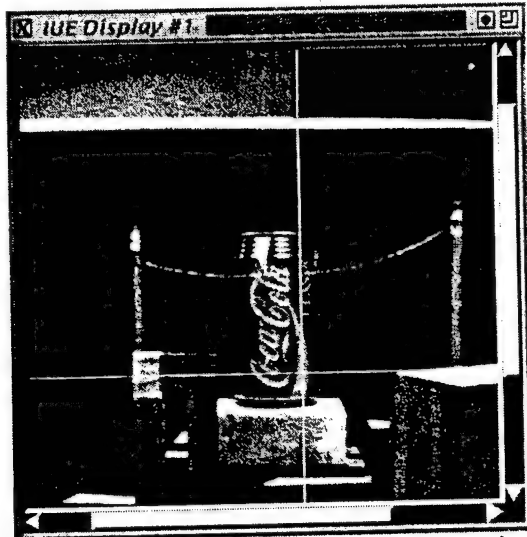


Figure 2.6. Interactive threshold selection.

In Figure 2.6, a cross-hair cursor is displayed to show the position when the user clicks on a point in the image (the intensity value is printed on the standard output of the application). After the user has pressed the `t` key, the intensity values greater than the selected threshold are mapped to red. This is shown in Figure 2.7.



Figure 2.7. Thresholded image.

2.2.1 Program Fragment

```
void
example2()
{
    DisplayObject  *obj1;
    ImageDouble    *im;
    char           name[50];

    /* display the intensity image in a DisplayObject window */
    obj1 = distrib->create_window(256, 256);
    cout << "please input PPM file name" << endl;
    cin >> name;
    im = distrib->load_image(name);
    obj1->set_clut(0, 256, 0, 0, 0, 255, 255, 255);
    obj1->set_intensity(im, 1.0, 1.0);
    obj1->display();

    /* set overlay buffer clut to display the thresholding overlay */
    obj1->set_buffer(1);
    obj1->set_clut(0, 2, 0, 0, 0, 255, 0, 0);
    obj1->set_crossHair();
}
```

2.3 ORB with Different Mapping Functions

To analyze the image in detail, the IUE provides a special tool called the Object Registered Browser (ORB). It serves as a magnifier on the image and it has a navigation control panel to aid the user in moving around the image. A field in the ORB corresponds to a pixel in the image. Subfields can be created to describe relationships between pixels.

Only a portion of an image can fit into the ORB. The following screen shots illustrate that the ORB is showing the area around point (266, 271). The current angle of rotation is negative (clock-wise) 40 degrees. The navigation control panel is shown in Figure 2.8.

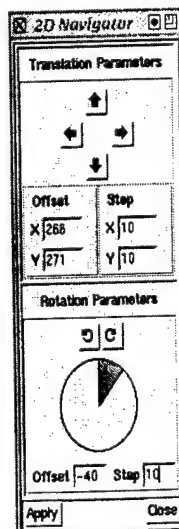


Figure 2.8. Navigation control panel.

The results of applying this mapping to the original image is shown in Figure 2.9.

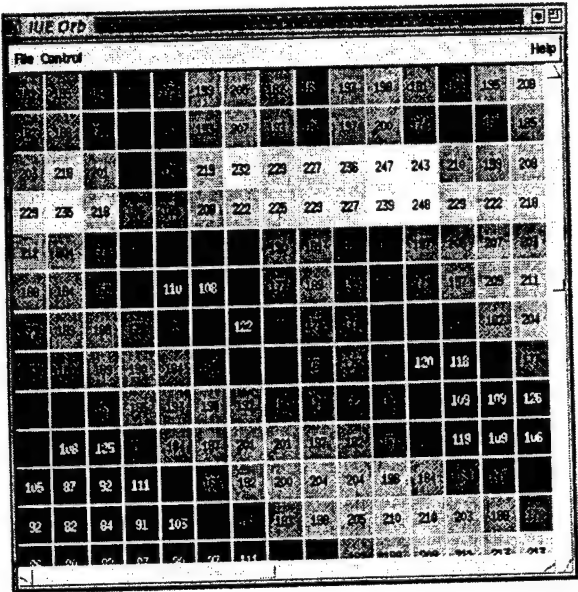


Figure 2.9. ORB for original image.

The results of applying this mapping to the darkened, lightened, interpolated, and colored images are shown in Figures 2.10 to 2.13, respectively.

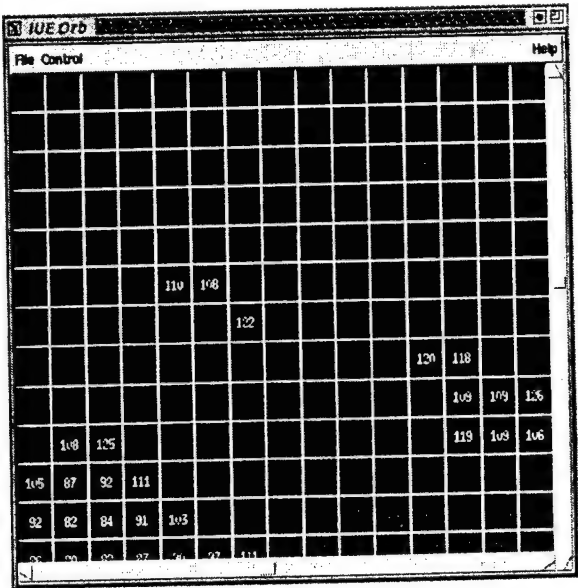


Figure 2.10. ORB for darkened image.

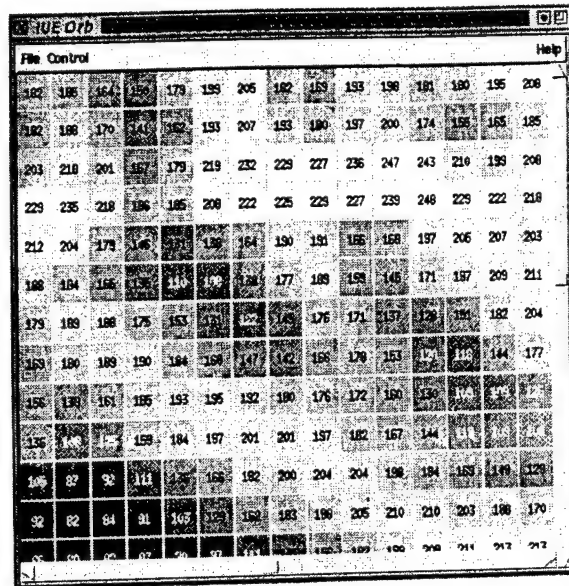


Figure 2.11. ORB for lightened image.

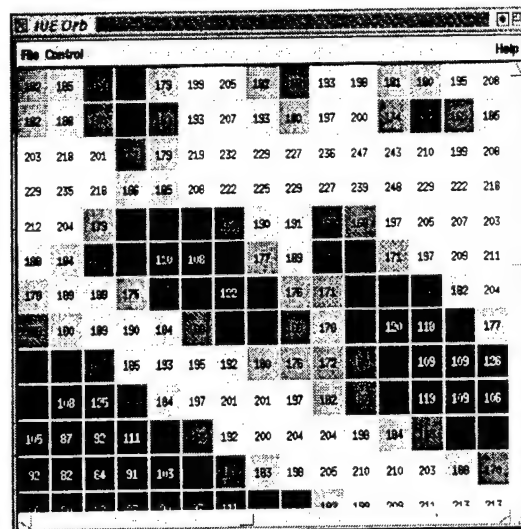


Figure 2.12. ORB for interpolated image.

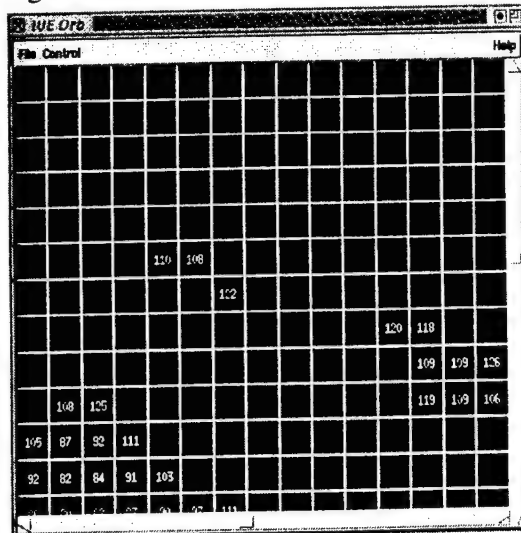


Figure 2.13. ORB for colored image.

2.3.1 Program Fragment

```
void
example3()
{
    Orb *orb;
    ImageDouble *im;
    char name[50];

    /*
     * Display image with default linear mapping parameters.
     * Map the range (0.0, 255.0) to 0 and 255.
     * Clut will linearly interpolate colors from (0, 0, 0) to (255, 255, 255)
     * for the 255 entries.
     */
    cout << "please input PPM file name" << endl;
    cin >> name;
    im = distrib->load_image(name);

    orb = distrib->create_orb();
    orb->intensity_orb(im, 266, 271, 15, 23);
    orb->showOrb();
    distrib->EventLoop();

    /*
     * Apply different mapping parameters on the
     * current buffer to make it darker
     */
    orb->set_subfield_mapping(0, 0, 0, 255, 0, 15);
    orb->display_orb();
    distrib->EventLoop();

    /*
     * Apply different mapping parameters on the
     * current buffer to make it lighter
     */
    orb->set_subfield_mapping(0, 0, 0, 255, 16, 31);
    orb->display_orb();
    distrib->EventLoop();

    /*
     * Apply different mapping parameters on the current buffer to
     * interpolate between 100.0 and 200.0.
     */
    orb->set_subfield_mapping(0, 0, 100, 200, 0, 31);
    orb->display_orb();
    distrib->EventLoop();

    /*
     * Change clut mapping
     */
    orb->set_clut(0, 8, 0, 0, 255, 0, 0, 0);
}
```

```

orb->set_clut(8, 7, 0, 0, 0, 255, 0, 0);
orb->set_subfield_mapping(0, 0, 0, 255, 0, 14);
orb->display_orb();
distrib->EventLoop();
}

```

2.4 Zooming with the ORB

DisplayObjects and the ORB object can be combined to show the same image from multiple perspectives. Figure 2.14 shows an image displayed using two DisplayObject windows. One window shows the original image, and the second window shows a zoomed image. The ORB window shows intensity values as text labels. The upper-left window shows the original image, and the bottom-left window shows the image with a zoom factor of 2 in both the X and Y directions.

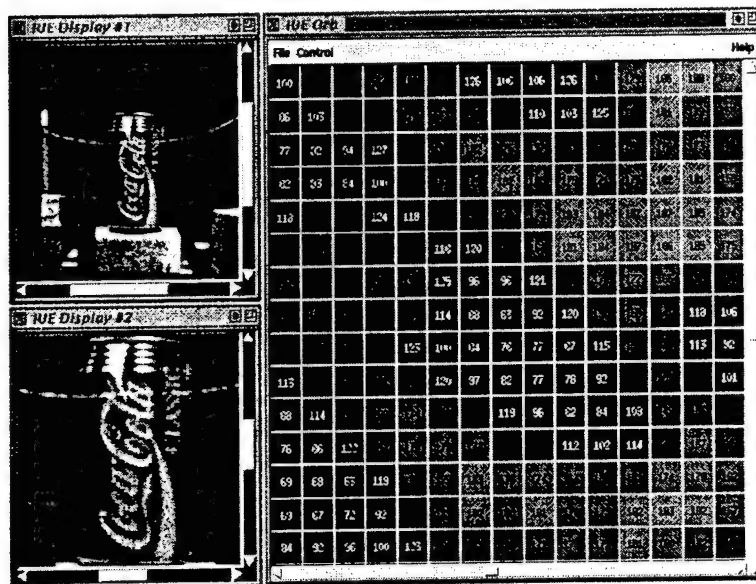


Figure 2.14. ORB zooming.

2.4.1 Program Fragment

```

void
example4()
{
    DisplayObject  *obj1, *obj2;
    ImageDouble    *im;
    char           name[50];

    /*
     * Creates a window with size 256 * 256.
     */
    obj1 = distrib->create_window(256, 256);

    /*
     * Input a ppm file name.  Read it and create an ImageObject.
     */
}

```

```

*/
cout << "please input PPM file name" << endl;
cin >> name;
im = distrib->load_image(name);

/*
 * Set a range of entries in the current buffer's clut with red green
 * and blue values interpolated between 0 0 0 and 255 255 255. The
 * range starts from 0 and extends to 255.
 */
obj1->set_clut(0, 256, 0, 0, 0, 255, 255, 255);

/*
 * Set the intensity image with a zoom size of 1.0 * 1.0 and
 * display it using the default mapping functions.
 */
obj1->set_intensity(im, 1.0, 1.0);
obj1->display();

/*
 * Display zoom image on obj2 (from obj1) with zoom factors
 * and a display offset
 */
PhotoImage      *block;
double           xz, yz;

obj2 = distrib->create_window(256, 256);

/*
 * Get the current photo image in window 1.
 */
block = obj1->get_photoImage();
cout << "please input zoom size -- xzoom and yzoom" << endl;
cin >> xz >> yz;

/*
 * Zoom the photoImage block by factor xz, yz. tkPhoto only
 * supports zooming by an integer factor. Display the zoomed image
 * from (0, 0) in the photo window. There is nothing in the intensity
 * buffer and the overlay buffer of window 2. This is the fastest way
 * to zoom something in a View Only window because zooming is done
 * in the Photo widget.
 */
obj2->display_photoImage(block, 0, 0, (int) xz, (int) yz);

/*
 * Display the same image in the ORB, in ORB_intensity (see orbrowse.C).
 * Here the number of fields in the ORB is 15 * 23. Translate (266, 271)
 * in image coordinates to the origin point (0, 0) in ORB coordinates.
 */
Orb          *orb;

```

```

orb = distrib->create_orb();
orb->intensity_orb(im, 266, 271, 15, 23);
orb->showOrb();
}

```

2.5 Edges Over the Image

One advantage of having subfields in the ORB is that one can bind different semantic information to different subfields. This example shows how subfields can be used for displaying edges over the image in the ORB. As a reference, edges also are provided over the image using the DisplayObject class. Two screen shots, which differ slightly from each other, are presented here. These screen shots differ in how the DisplayObjects perform the edge zooming. In Figure 2.15, the image is zoomed using the photo widget directly. This method is fast, but since the Photo Widget treats every pixel equally, the edge pixels get zoomed along with the intensity pixels, making the edges look thicker.

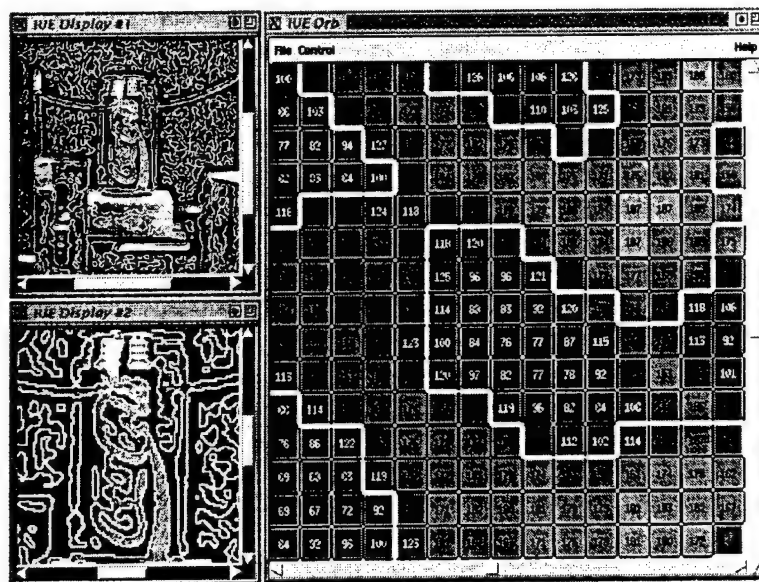


Figure 2.15. Zooming with the Photo Widget.

In Figure 2.16, the application has knowledge of the edge information. Those pixels on the edges are not zoomed, so that the edges appear to be thinner and their relative positions appear to be more accurate.

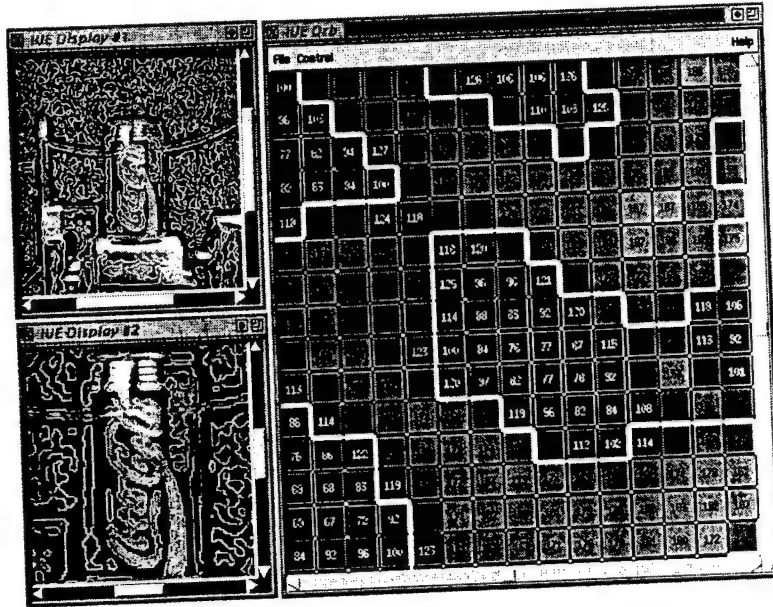


Figure 2.16. Application-based zooming.

2.5.1 Program Fragment

```
void
example5()
{
    DisplayObject *obj1, *obj2;
    ImageDouble *im, *gaussian, *laplacian;
    ImageVector2 *imVec;
    char name[50];

    /*
     * display zero crossings at the original size
     */
    obj1 = distrib->create_window(256, 256);
    cout << "please input PPM file name" << endl;
    cin >> name;
    im = distrib->load_image(name);
    obj1->set_clut(0, 256, 0, 0, 0, 255, 255, 255);
    obj1->set_intensity(im, 1.0, 1.0);
```

```

imVec = im->build_gradient2X2();
gaussian = im->gaussian_convolve(2.0, 0.01);
laplacian = gaussian->build_laplacian();

obj1->set_buffer(1);
obj1->set_clut(0, 2, 0, 0, 0, 255, 0, 0);
obj1->set_zCross(laplacian, imVec, 1.0, 1.0, 0.0);
obj1->display();

/*
 * display zero crossings on a zoomed window with input zoom size
 */
double          xz, yz;

cout << "please input zoom size -- xzoom and yzoom" << endl;
cin >> xz >> yz;

obj2 = distrib->create_window(256, 256);
obj2->set_image(im);
obj2->set_clut(0, 256, 0, 0, 0, 255, 255, 255);
obj2->set_intensity(im, xz, yz);

obj2->set_buffer(1);
obj2->set_clut(0, 2, 0, 0, 0, 255, 0, 0);
obj2->set_zCross(laplacian, imVec, xz, yz, 0.0);
obj2->display();

/*
 * display zero crossings with the ORB
 */
Orb          *orb;

orb = distrib->create_orb();
orb->zeroCross_orb(im, laplacian, 266, 271, 15, 23);
orb->showOrb();
}

```

2.6 Laplacian

Determining the location of contours is easier when Laplacian images are displayed accompanied by overlaid edgel images. In Figure 2.17, the edge lines divide the image into separate areas: red areas have positive intensity values, blue areas have negative intensity values.

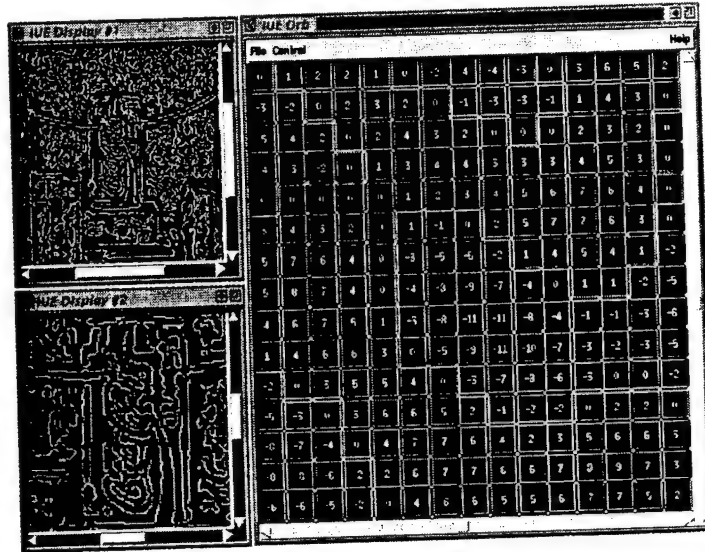


Figure 2.17. Laplacian image.

2.6.1 Program Fragment

```
void
example6()
{
    DisplayObject *obj1, *obj2;
    ImageDouble   *im, *gaussian, *laplacian;
    ImageVector2  *imVec;
    char          name[50];

    /*
     * Display zero crossings at the original size.
     */
    obj1 = distrib->create_window(256, 256);
    cout << "please input PPM file name" << endl;
    cin >> name;
    im = distrib->load_image(name);
    imVec = im->build_gradient2X2();
    gaussian = im->gaussian_convolve(2.0, 0.01);
    laplacian = gaussian->build_laplacian();
    delete gaussian; /* free memory space */

    obj1->set_image(laplacian);
    obj1->set_clut(0, 256, 0, 0, 0, 255, 255, 255);
    obj1->set_intensity(laplacian, 1.0, 1.0);
    obj1->set_mapping(laplacian->minval, laplacian->maxval, 0, 255);
}
```



```

obj1->set_buffer(1);
obj1->set_clut(0, 2, 0, 0, 0, 255, 0, 0);
obj1->set_zCross(laplacian, imVec, 1.0, 1.0, 0.0);
obj1->display();

/*
 * Display zero crossings on a zoomed window with the
 * zoom size specified by the user.
 */
double          xz, yz;

cout << "please input zoom size -- xzoom and yzoom" << endl;
cin >> xz >> yz;

obj2 = distrib->create_window(256, 256);
obj2->set_image(laplacian);
obj2->set_clut(0, 256, 100, 100, 100, 255, 255, 255);
obj2->set_intensity(laplacian, xz, yz);
obj2->set_mapping(laplacian->minval, laplacian->maxval, 0, 255);

obj2->set_buffer(1);
obj2->set_clut(0, 2, 0, 0, 0, 255, 0, 0);
obj2->set_zCross(laplacian, imVec, xz, yz, 0.0);
obj2->display();

/*
 * Display zero crossings on the ORB.
 */
Orb          *orb;

orb = distrib->create_orb();
orb->zeroCross_orb(laplacian, laplacian, 266, 271, 15, 23);
orb->set_clut(0, 8, 0, 0, 255, 0, 0, 0);
orb->set_clut(7, 8, 0, 0, 0, 255, 0, 0);
orb->set_clut(16, 2, 0, 0, 0, 0, 255, 0);
orb->set_subfield_mapping(0, 0, -8, 8, 0, 14);
orb->set_subfield_mapping(0, 1, 0, 1, 16, 17);
orb->set_subfield_mapping(1, 0, 0, 1, 16, 17);
orb->showOrb();
}

```

2.7 Links

Linking the DisplayObjects with ORB objects can provide for better navigation. The following example is a simple application of linking. A DisplayObject displays a rectangular box on top of the image to show what the ORB object is displaying in its window. When the user changes one end of the link, for example, resizes the ORB window or pans the ORB display, the other end of the link updates too, so the rectangular box adjusts according to the ORB window changes.

Figure 2.18 shows the rectangular box on top of the image, and Figure 2.19 shows the associated ORB canvas.



Figure 2.18. Linked DisplayObject window.

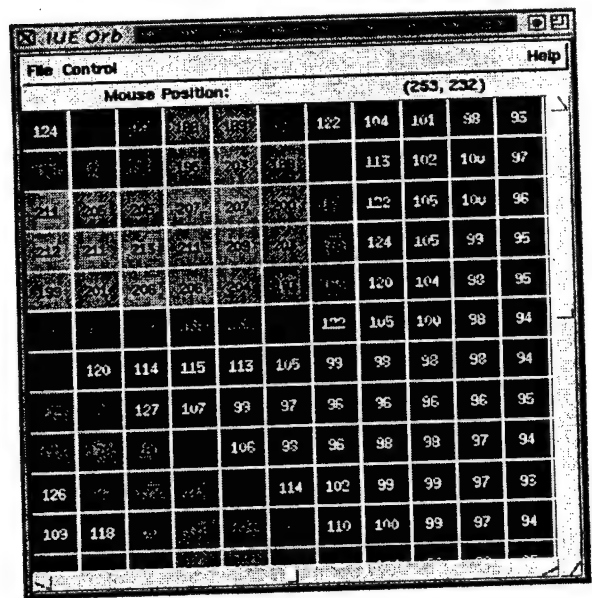


Figure 2.19. Linked ORB canvas.

2.7.1 Program Fragment

```
void
example7()
{
    DisplayObject *obj1;
    Orb *orb;
    ImageDouble *im;
    char name[50];

    obj1 = distrib->create_window(256, 256);
    cout << "please input PPM file name" << endl;
    cin >> name;
    im = distrib->load_image(name);
    obj1->set_clut(0, 256, 0, 0, 0, 255, 255, 255);
    obj1->set_intensity(im, 1.0, 1.0);
    obj1->display();

    orb = distrib->create_orb();
    orb->intensity_orb(im, 200, 100, 10, 10);

    /* Link ORB with obj1 using display_square_void() as a link function.
     * Set up the link parameters and link functions.
     */

    LinkArgvs *argvs;

    argvs = new LinkArgvs;
    argvs->xOffset = &(orb->hOrbOffset);
    argvs->yOffset = &(orb->vOrbOffset);
    argvs->width   = &(orb->hFieldNum);
    argvs->height  = &(orb->vFieldNum);
    argvs->xOrigin = &(orb->hOrbOrigin);
    argvs->yOrigin = &(orb->vOrbOrigin);

    orb->set_link(obj1,
        DisplayObject::display_square_void,
        (void *)argvs);

    orb->showOrb();
}
```

2.8 Snake

Snakes are dynamic contours that are able to iteratively conform to contours in images. This example illustrates the computation of a snake. The user is able to control the initial points using an ORB object, and the forces associated with each node can be visualized as the snake converges.

Figure 2.20 shows the DisplayObject window, and Figure 2.21 shows the associated ORB canvas. The lower half of the pencil eraser is displayed in the ORB window.

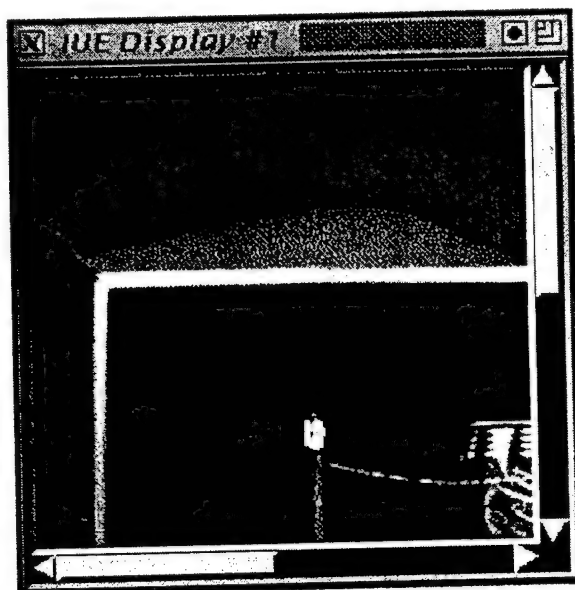


Figure 2.20. Snake DisplayObject window.

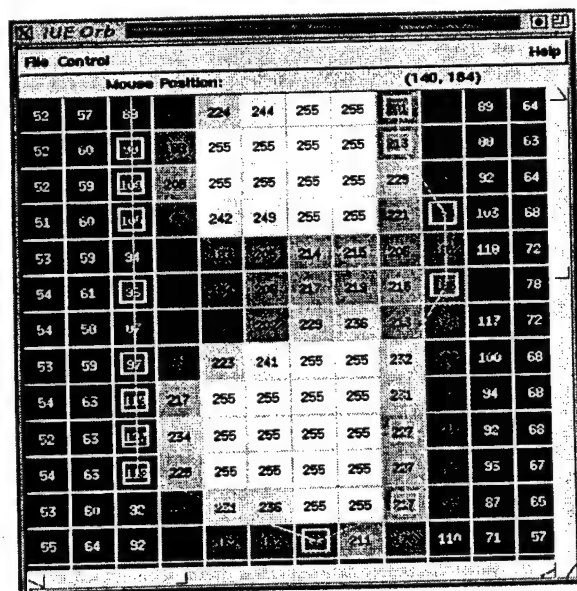


Figure 2.21. Snake ORB canvas.

2.8.1 Program Fragment

```
void
example8()
{
    DisplayObject *obj1;
    ImageDouble   *im;
    ImageVector2  *imVec;
    char          name[50];

    /*
     * Creates a window with size 256 * 256.
     */
    obj1 = distrib->create_window(256, 256);

    /*
     * load a ppm image file
     */
    cout << "please input PPM file name" << endl;
    cin >> name;
    im = distrib->load_image(name);

    /*
     * Set current intensity buffer's clut.
     */
    obj1->set_clut(0, 256, 0, 0, 0, 255, 255, 255);
    obj1->set_intensity(im, 1.0, 1.0);
    obj1->display();

    /*
     * Creates gradient image imVec on im.
     */
    imVec = im->build_gradient2X2();
    obj1->set_image(imVec);

    /*
     * Create an intensity ORB which shows the intensity image im.
     * The offset is 140, 188. i.e the value at (140, 188) in
     * image coordinates is mapped to the ORB origin.
     */
    Orb *orb;

    orb = distrib->create_orb();
    orb->intensity_orb(im, 140, 188, 15, 23);
    orb->showOrb();

    /*
     * Bind event functions and event keys for running the snake
     * and deleting snake points by:
     * distrib->bind_event(evKey, evFunc, funcName, funcInfo, evId).
     * If you don't know which evId you should put to insert this
     * new event without conflicting with other existing event bindings,

```

```

* you just set evId to -1. If you set evId to -2, it will try
* to find that evKey in an event table and add new attributes.
* Otherwise, the evId will be a number from 1 to 100.
*
* These bindings can also be set up as default bindings. See
* Distributor::associate_default_events() in file tkdistrib.C.
*/

distrib->bind_event("s", &Distributor: :EV_snake,
"sake", "dynamic snake", -1);
distrib->bind_event("S", &Distributor: :EV_deleteSnake,
"deletesnake", "delete dynamic snake", -1);

/*
* Set the overlay Mask on, so as you click a point in the ORB,
* it will show the mask for points you clicked.
*/
orb->set_overlayMask();

cout << endl
<< "Please click initial snake points on photo image or orb" << endl;
}

/* Distributor::EV_snake() is in file "tkdistrib.C" */
Distributor::EV_snake(char *)
{
    List<Point> *snakePt;
    int threshold;

    snakePt = get_ptlist();
    cout << " Please input balance pixel distance " << endl;
    cin >> threshold;

    dynamic_snake(currentDisplayObject,
        currentObjectRegisterBrowser,
        currentDisplayObject->get_imageVector(),
        snakePt,
        threshold);
}

/* ::dynamic_snake() is in file "driver.C" */
void
::dynamic_snake(DisplayObject *obj, Orb *orb, ImageVector2 *imVec,
List<Point> *ptList, int threshold)
{
    int i, j, k, ind, n;
    PhotoImage *recoverPixels;
    Point neighbor[9] = {{0, -1}, {1, -1}, {1, 0}, {1, 1},
        {0, 1}, {-1, 1}, {-1, 0}, {-1, -1}, {0, 0}};
    double dd;
    double energy;
    int Poptimal;

```

```

double  Eoptimal, Eold;
double  energyMtx[300][9];
int  posMtx[300][9];
Point   *currNode, *nextNode;
Point   v1, v2;

n = ptList->next() - 1;
recoverPixels = obj->draw_snake(ptList);
if (orb != NULL)
orb->set_ptlist(ptList);

for (i = 0; i < 9; i++) {
    energyMtx[0][i] = 0;
    posMtx[0][i] = 8;
}

Eoptimal = 1000000000.0;
int frameCount = 0;
do {
    Eold = Eoptimal;
    for (i = 1; i < n; i++) {
currNode = ptList->find(i);
nextNode = ptList->find(i + 1);
for (j = 0; j < 9; j++) { /* for next node */
    energyMtx[i][j] = 1000000000.0;
    posMtx[i][j] = 8;
    v2.x = nextNode->x + neighbor[j].x;
    v2.y = nextNode->y + neighbor[j].y;
    for (k = 0; k < 9; k++) { /* for current node */
        v1.x = currNode->x + neighbor[k].x;
        v1.y = currNode->y + neighbor[k].y;
        dd = dist(&v1, &v2);
        if (dd == 0) energy = 1000000000.0;
        else energy = energyMtx[i-1][k]
+ fabs(dd - threshold) * 10.0
- energy_image(imVec->get_val(v1.x, v1.y));
        if (energy < energyMtx[i][j]) {
            energyMtx[i][j] = energy;
            posMtx[i][j] = k;
        }
    }
}
}

/* find optimal energy */
Eoptimal = energyMtx[n-1][8];
Poptimal = 8;
for (i = 0; i < 8; i++)
if (energyMtx[n-1][i] < Eoptimal) {
    Eoptimal = energyMtx[n-1][i];
    Poptimal = i;
}

```

```

obj->recover_snake(ptList, recoverPixels);

/* move snake to new positions */
ind = Poptimal;
for (i = n; i > 0; i--) {
currNode = ptList->find(i);
currNode->x += neighbor[ind].x;
currNode->y += neighbor[ind].y;
ind = posMtx[i - 1][ind];
}
recoverPixels = obj->draw_snake(ptList);

if (orb != NULL)
orb->display_orb();
    Tcl_Eval(interp, "update");
} while (Eold != Eoptimal);
}

```

2.9 Graph Browser

This phase of research began by prototyping many different parts of the user interface to complete the functional specification, and to answer basic implementation questions about choices regarding GUIs and user interface toolkits. Prototyping of the interface has gone through three distinct phases. First mock-ups of the different interface objects were developed using the Interface Builder on the NeXT machine. This allowed for rapidly prototyping the objects for look-and-feel. For reasons of rapid development, implementations next were started in C on Silicon Graphics (SGI) machines using the GL graphics library, Motif, and the FORMS user interface toolkit. Using these, the general display object and the different browsers were put up very quickly. As part of this, extensions to GNUPlot were explored to ensure its compatibility with the methods associated with the general display class and that it could provide an inexpensive plotting package. OPENGL was also evaluated as a possible machine independent graphics package to give users the powerful functionality of the SGI graphics library and FORMS. In the third stage of prototyping, implementation of the IUE objects in Tk/Tcl in C++ began. This provides a very rich machine independent tool-kit for developing interactive user interfaces.

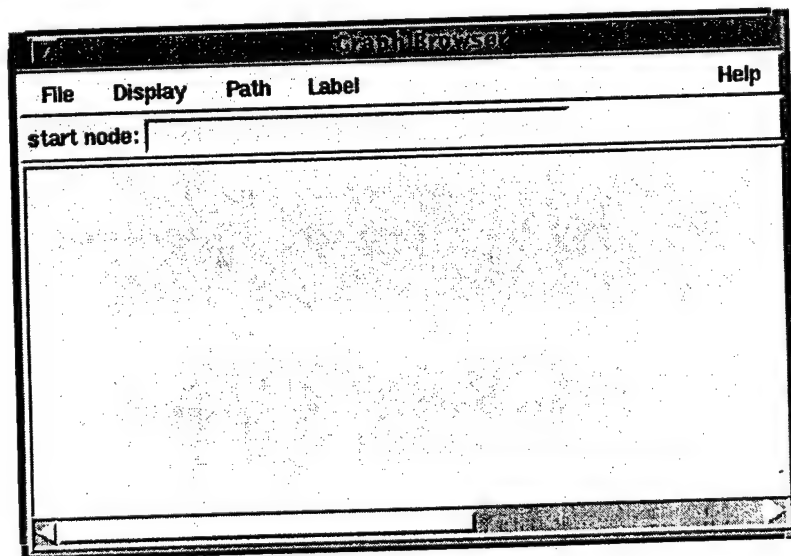


Figure 2.22. Initialized graph browser.

Figures 2.22 - 2.30 show the Tk/Tcl C++ implementation of the graph browser.

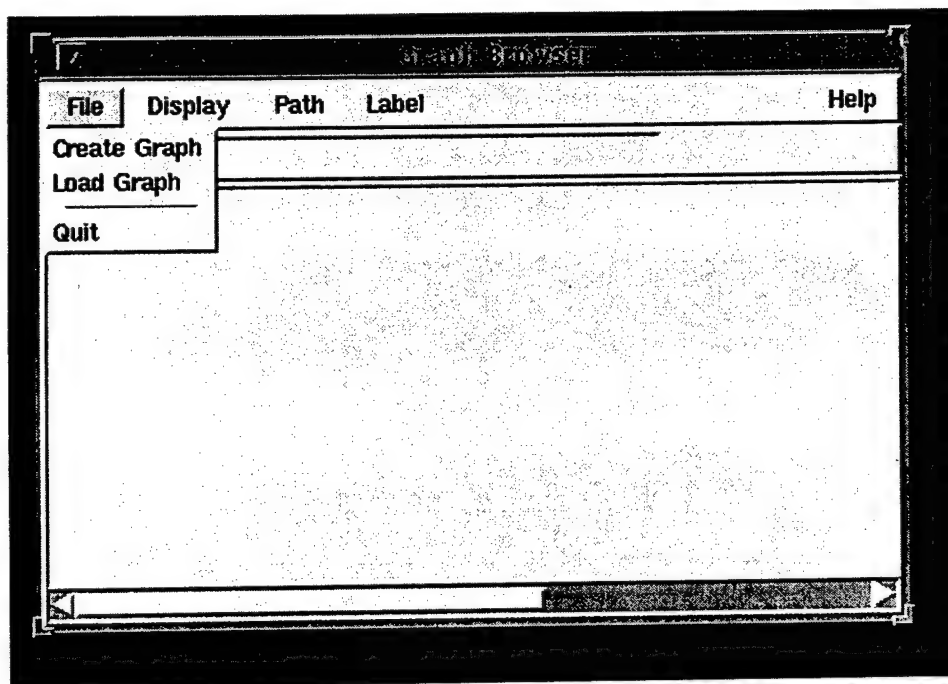


Figure 2.23. Creating a graph.

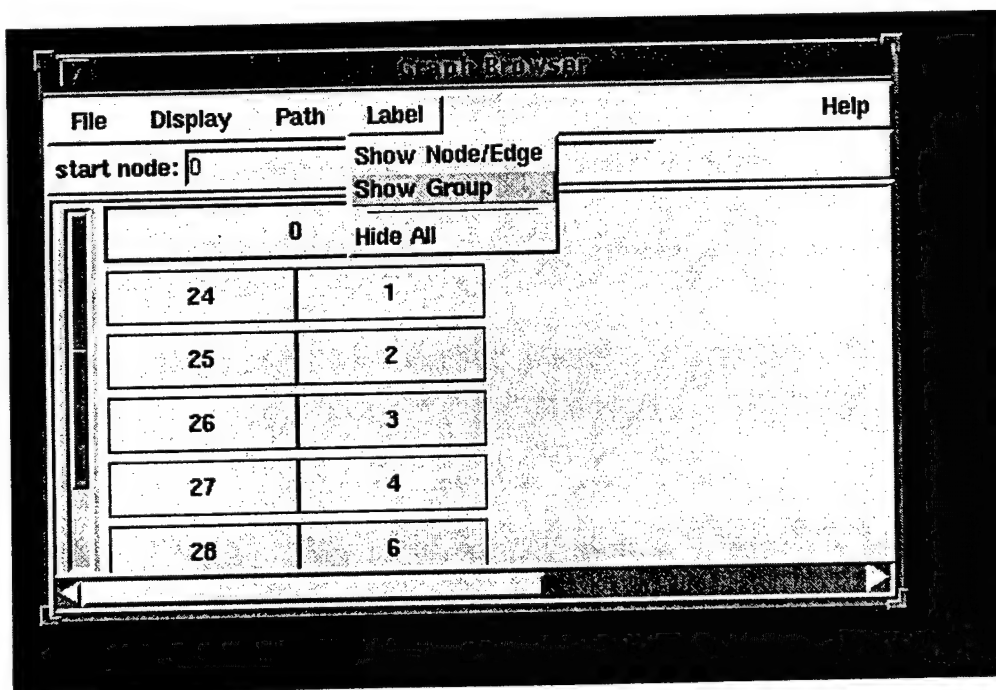


Figure 2.24. Searching a node

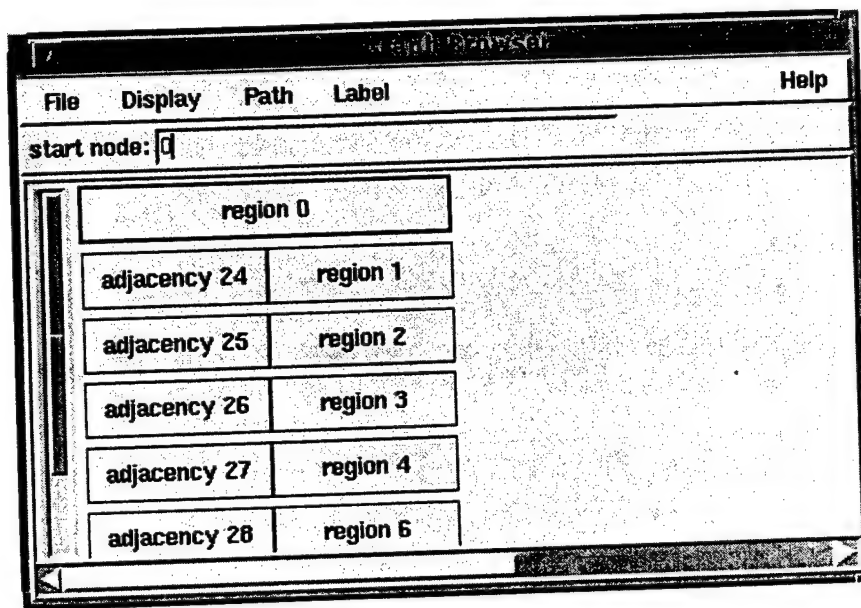


Figure 2.25. Links from node 0.

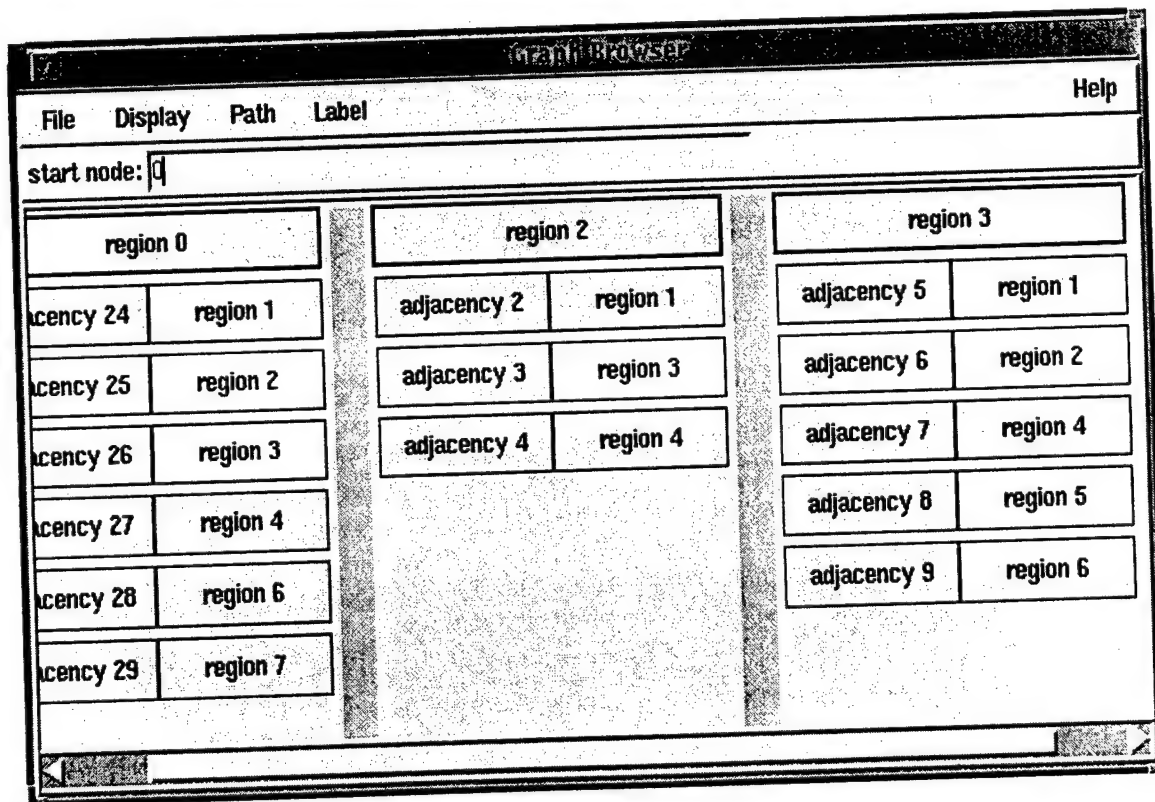


Figure 2.26. Traversing a path of the graph.

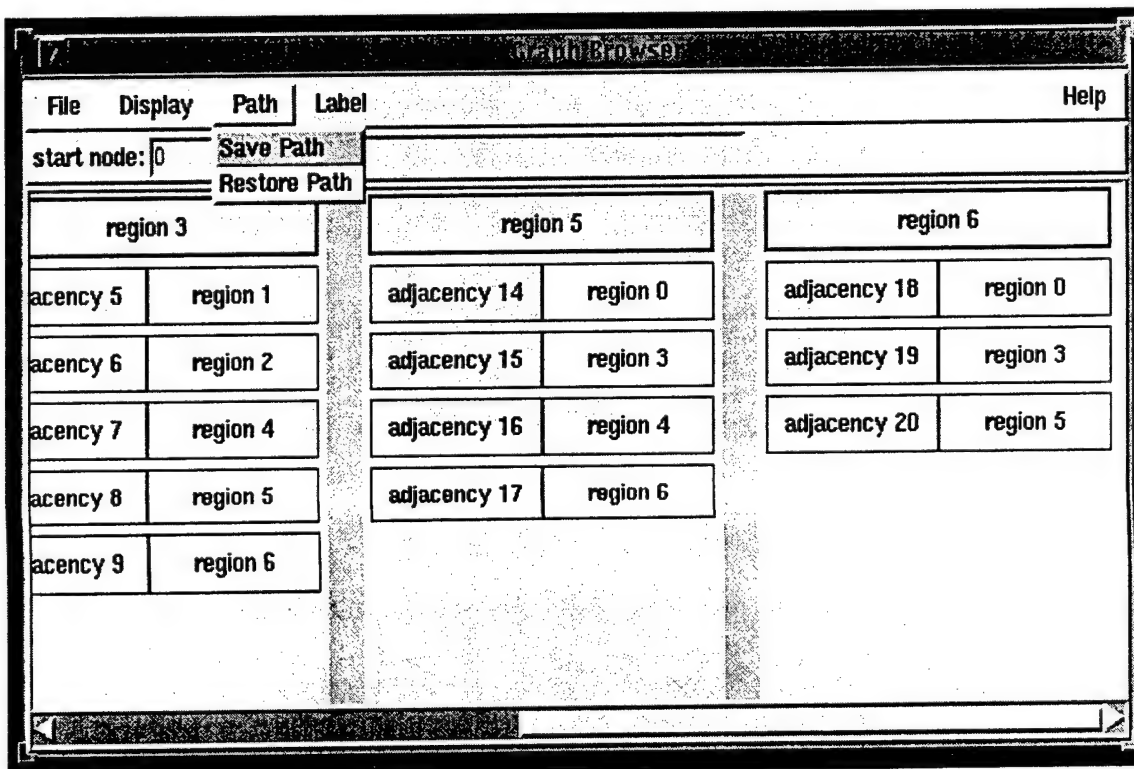


Figure 2.27. Saving a path of the graph.

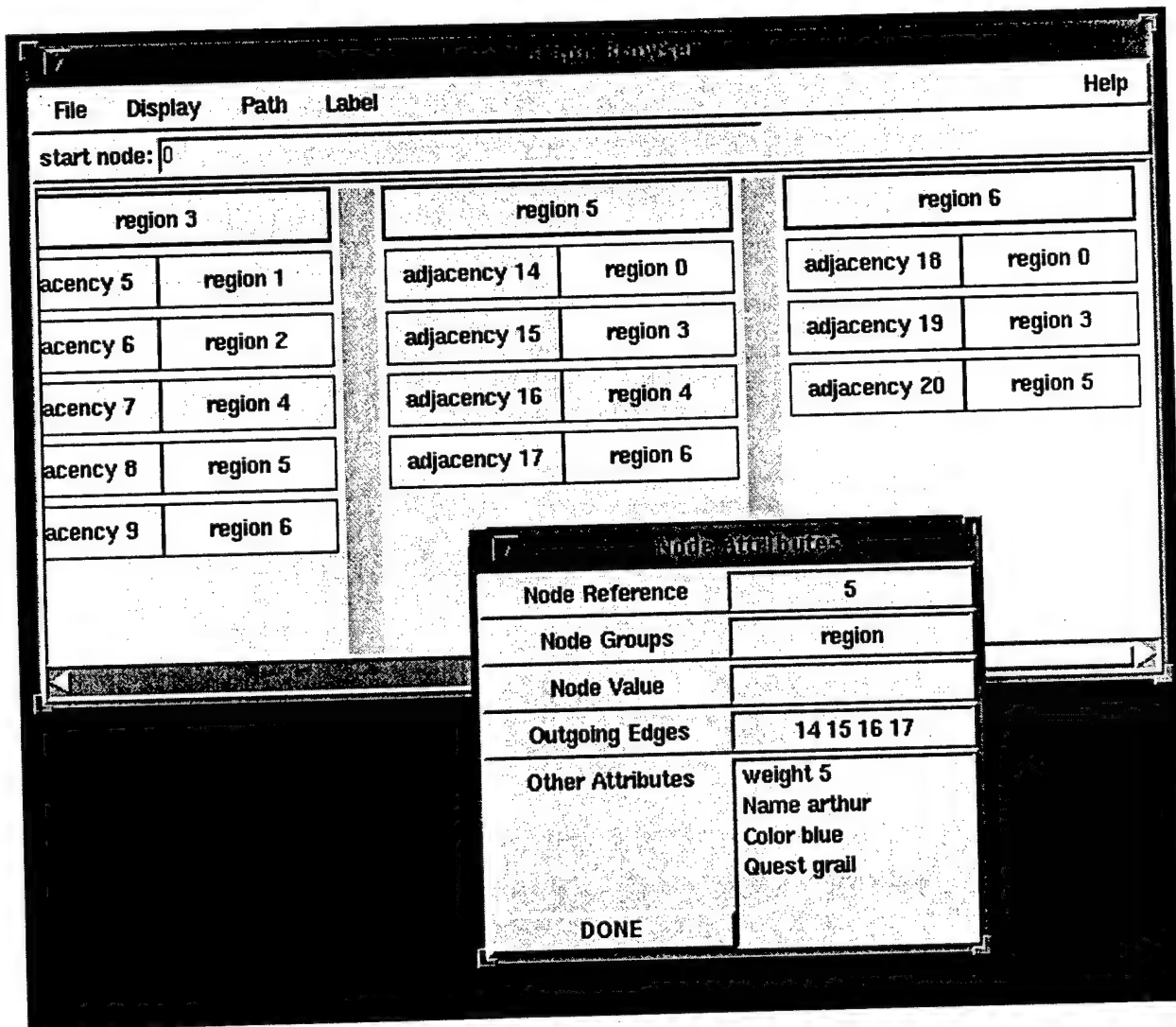


Figure 2.28. Inspecting node attributes.

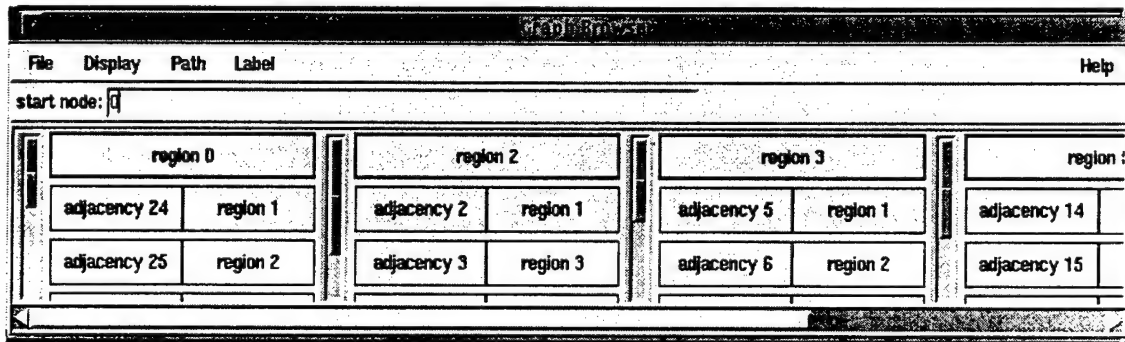


Figure 2.29. Inspecting graph path.

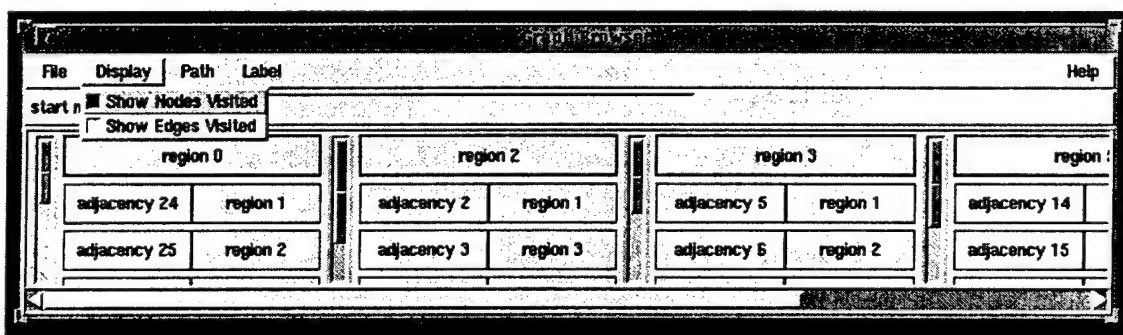


Figure 2.30. Showing visited nodes.

Chapter 3

World Wide Web Tools

This chapter describes several image understanding environment tools that were created for use via the WWW.

3.1 WWW Image Database

The WWW image database is a distributed image database written in HTML and freeWAIS. Each image in the database has a header containing information about the image as well as an image icon and a link to the image. These image headers are all stored at a single location (currently a computer at Georgia Tech called moralforce), but the actual images are located at various sites throughout the network.

The image database can be accessed using a publicly available WWW browser such as Mosaic or Netscape. The image database search engine is based on freeWAIS, so the Mosaic or Netscape browser must have been compiled with WAIS support. The HTTP address for the image database is <http://moralforce.cc.gatech.edu>.

The image database contains a browser page and an image submission page. The image database browser page is shown in Figure 3.1. The database browser contains a box where the user can enter database queries. These queries are boolean combinations of keywords and the boolean operators 'and' and 'not'.

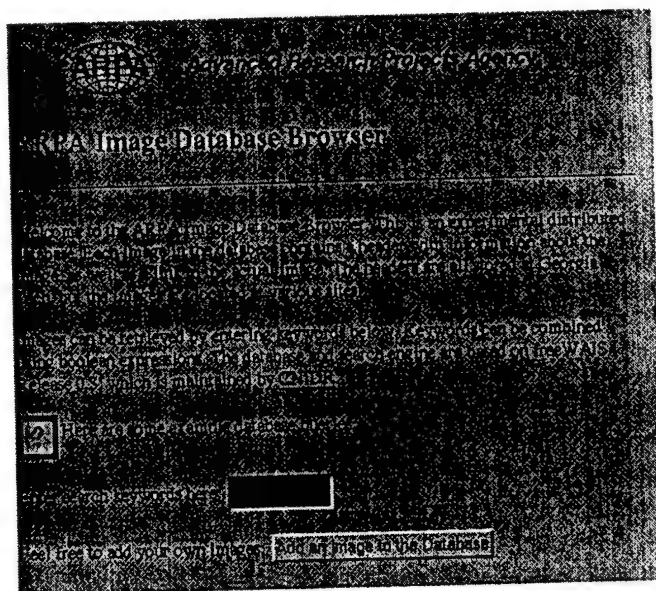


Figure 3.1. Image database browser.

A successful search results in the display of an image header. Figure 3.2 shows an example of a typical image header. The header contains information about the image and also about any publication in which the image had appeared. Notice that in this header some of the information ('Submitted by' and 'Organization Name') are actually links to other HTTP addresses. The header also contains an icon of the image, and a link to the actual image. If the user chooses to retrieve the image, then clicking on the link will cause the image to be down-loaded to the user's site.

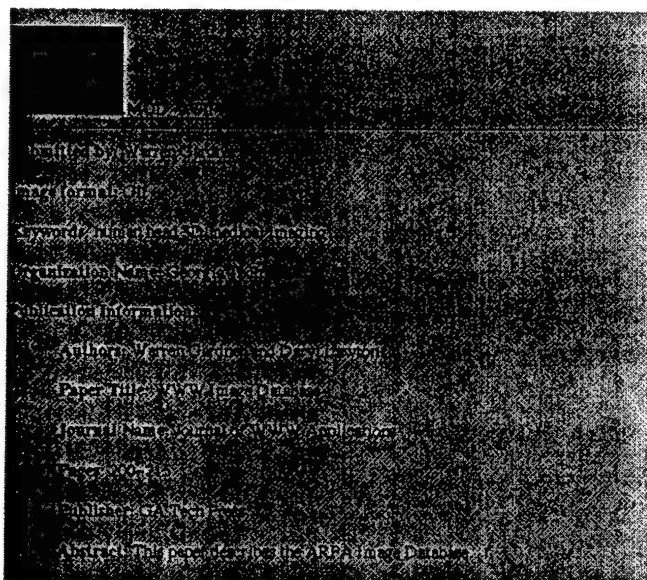


Figure 3.2. Image header.

The browser page also contains a button entitled 'Add an Image to the Database'. Clicking on this button brings up the image submission page.

This page is shown in Figures 3.3 and 3.4.

ARPA Advanced Research Projects Agency

Add an Image to the ARPA Database

Required Image Fields

Submitted By

Image Format

URL

Keywords

Optional Image Fields

Organization Name

Publication Information

Affiliation

Figure 3.3. Image submission page (top).

Author

Paper Title

Journal Name

Conference Name

Pages

Publisher

Abstract

Adding the Image

Once the fields have been filled in, click on the **ADD IMAGE** button to add the image to the database. If you wish to erase all the field entries and start with a new form, then press the **REFRESH** button.

Add Image Click this button to add the image.

Refresh Click this button if you do NOT want to add the image.

Figure 3.4. Image submission page (bottom).

The submission page contains several fields for the user to enter the appropriate information. Some of the fields are required, and the user cannot submit an image without filling in these fields. Once the user has entered the appropriate information, clicking on the 'Add Image' button on the bottom of the page will submit the image. The header is then constructed, and the database is reindexed. Currently anyone can enter an image, but access to the submission page can easily be restricted to a set of users.

3.2 Proceedings of the 1994 DARPA Image Understanding Workshop

The proceedings of the 1994 DARPA Image Understanding Workshop were reproduced in HTML format making these proceedings available electronically. The HTTP address for these proceedings is <http://moralforce.cc.gatech.edu/iuw.html>. The home page of the proceedings is shown in Figure 3.5.

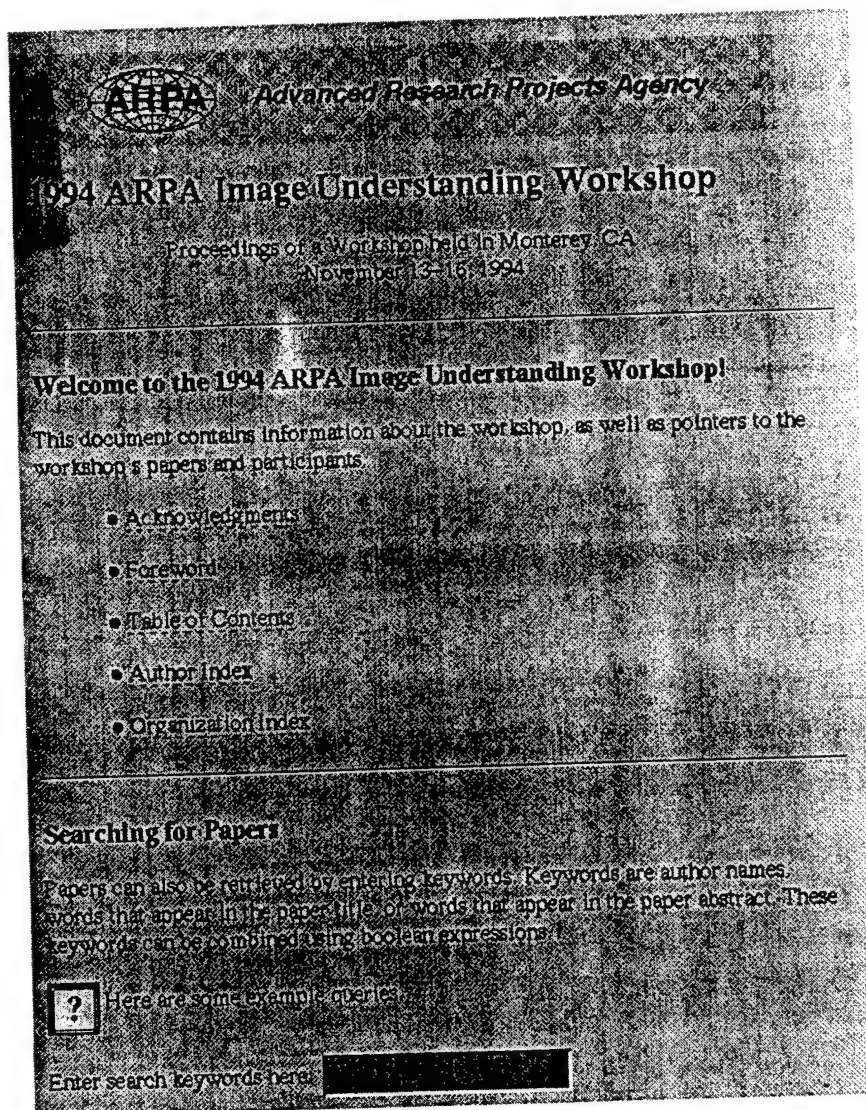


Figure 3.5. Proceedings of the 1994 DARPA IUW home page.

The electronic version of the proceedings contains acknowledgments, a foreword, a table of contents, an author index, and an organization index. The table of contents, author index, and organization index pages are shown in Figures 3.6 - 3.8.


 **Advanced Research Projects Agency**


Table of Contents

Volume II

Section I - Principal Investigator Reports

- 1. Image Understanding Research at Old Riverside
Bh. Bhanu
- 2. Image Understanding at MIT
John J. Vanilla, Aloimenes Kanti Chellappa, Larry S. Davis, and Azriel Rosenfeld
- 3. Image Understanding Research at Columbia University
Peter K. Allen, Terrence E. Bobit, John R. Kender, and Shree K. Nayar
- 4. Progress in Image Understanding at MIT
Walter Rumlton, B. G. Horn, and M. Perrio

Figure 3.6. Table of contents.

 **Advanced Research Projects Agency**

Author Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Alfred A. Abell

- From Blending to Words: Generating Qualitative Descriptions of Objects from Images

Steven A. Adami

- Computing Sweep Volumes for Sensor Planning Tasks
- CAD Model Acquisition Using Binary Space Partitioning Trees

Narendra Ahuja

- Multiscale Image Segmentation Using a Recent Transform
- QUATUQ: AN OVERVIEW OF RESEARCH DURING 1993-94
- Performance Analysis of Depth Cues for Active Vision
- SILHOUETTE-BASED STRUCTURE AND MOTION ESTIMATION OF A SMOOTH OBJECT
- Obtaining Focused Images Using a Non-frontal Imaging Camera

Figure 3.7. Author index.

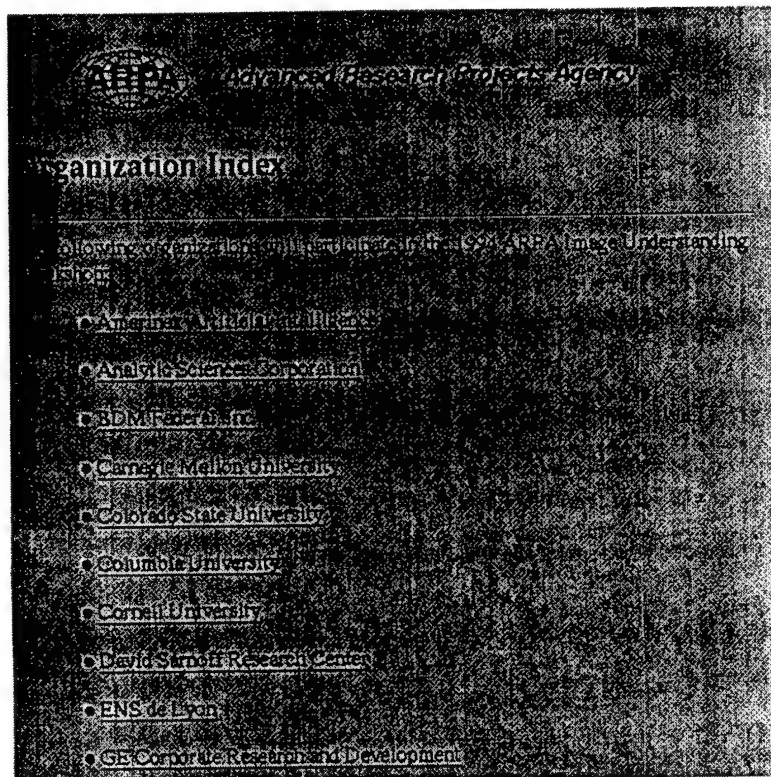


Figure 3.8. Organization index.

Each of these pages also contains links to author's home pages, organization home pages, and abstract pages. An example organization home page is shown in Figure 3.9.

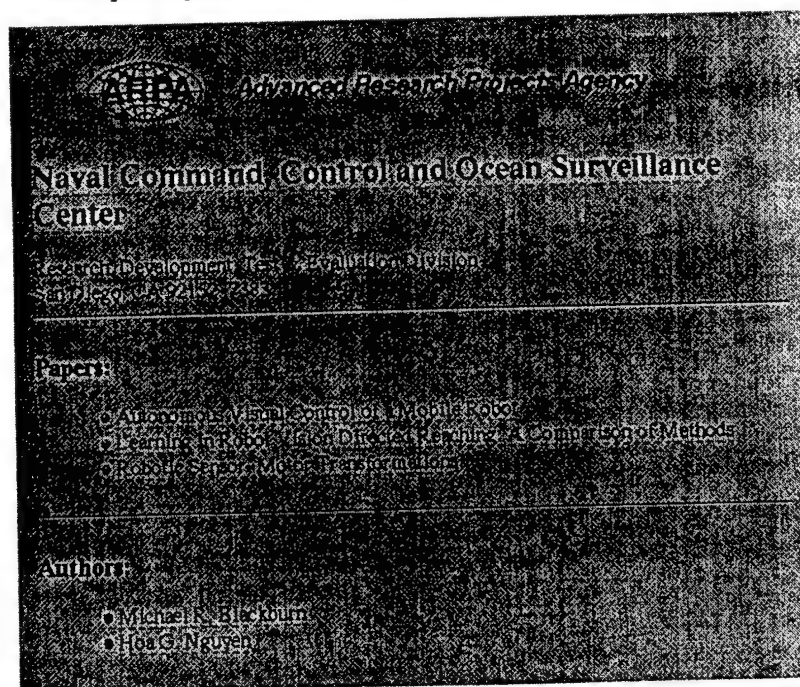


Figure 3.9. Organization home page.

Each organization with papers in the proceedings has an organization home page. This page contains a list of papers that authors affiliated with the organization have published in this proceedings.

An example abstract page is shown in Figure3.10.

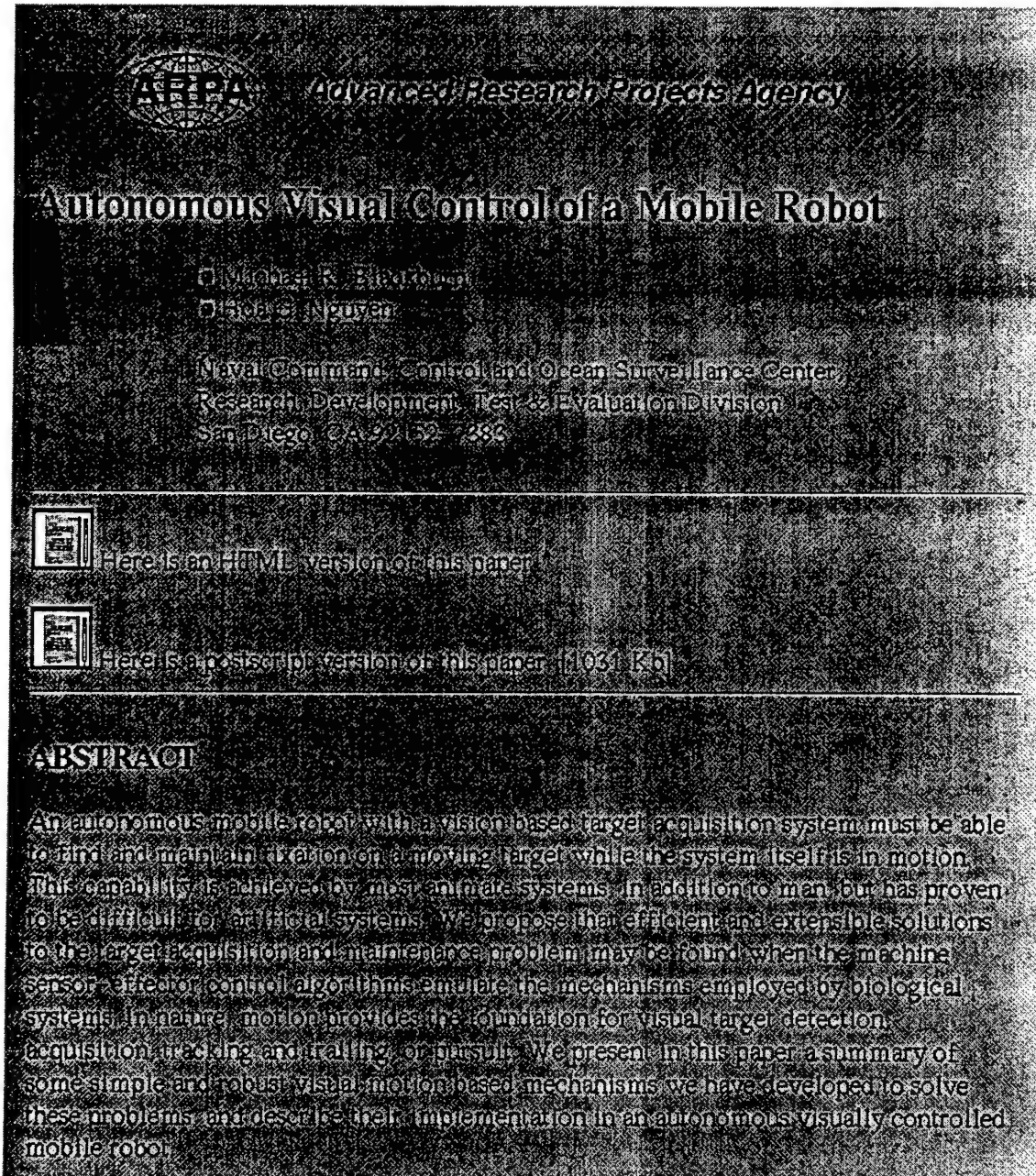


Figure 3.10. Paper abstract.

Each paper in the proceedings has an associated abstract page. This abstract page contains an abstract of the paper, as well as links to the paper's author and organization home pages. There also are links to postscript and HTML versions of the paper. The abstract pages can be searched using keywords such as subject areas or author names.

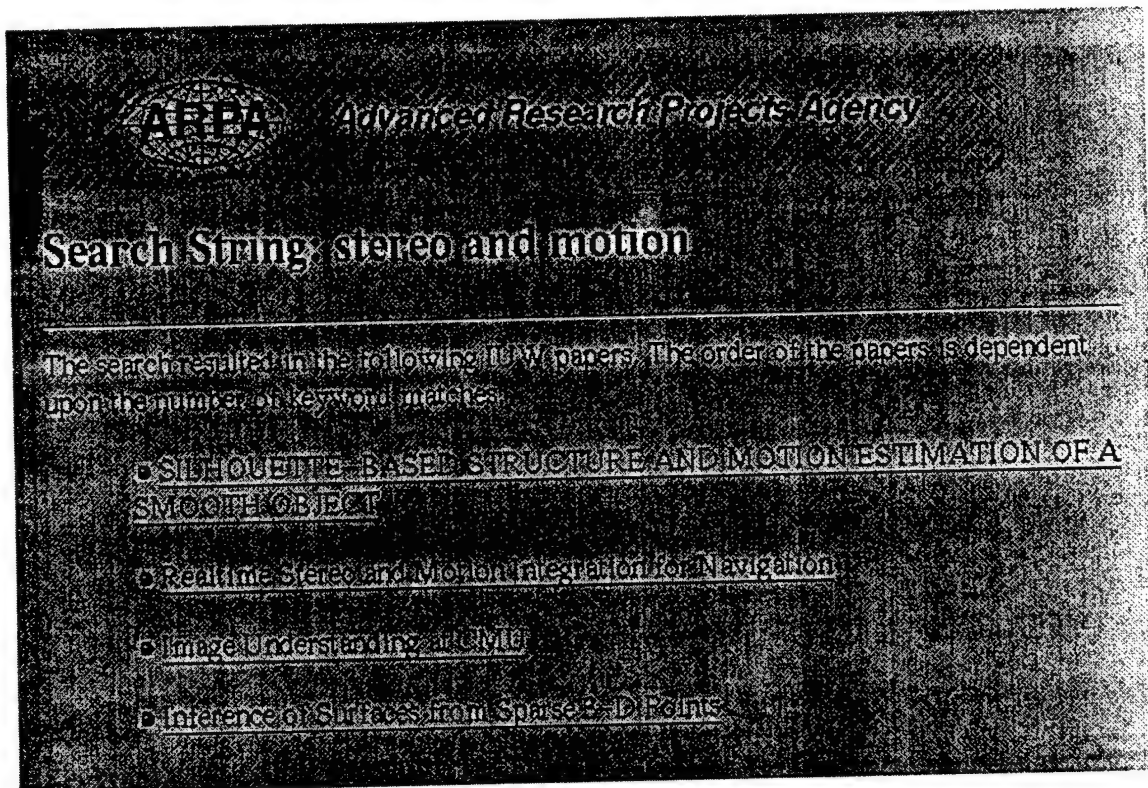


Figure 3.11. Search result.

The IUW home page shown in Figure 3.5 contains a box where the user can enter queries. These queries are boolean combinations of keywords and the boolean operators 'and' and 'not'. The result of searching for abstracts that contain the keywords 'stereo and motion' is shown in Figure 3.11. A CD-ROM version of the workshop proceedings was also produced and was distributed at the workshop. This contains all the papers included on the web version.

3.3 Snakes Tutorial

The snakes home page is shown in Figure 3.12

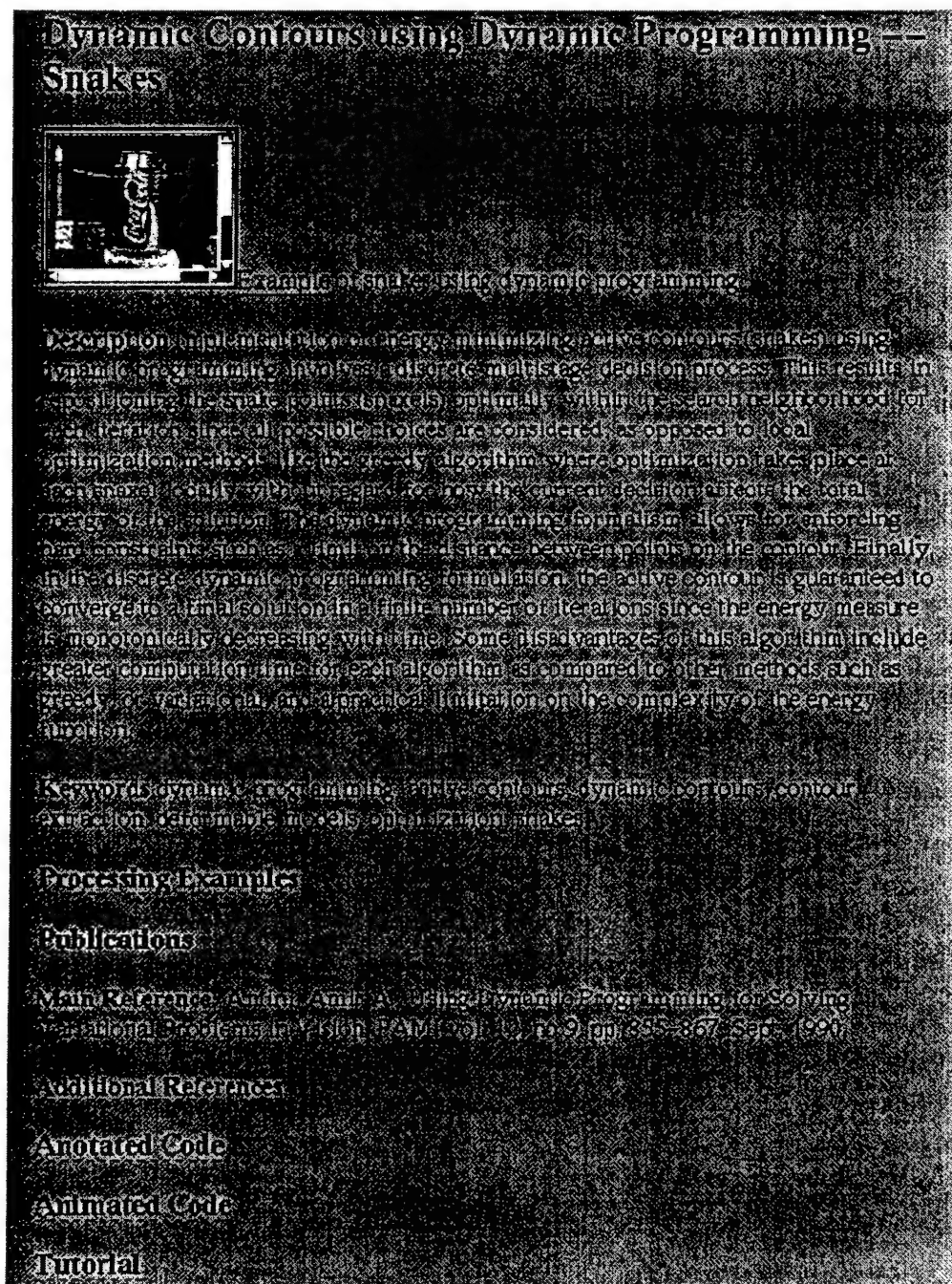


Figure 3.12. Snakes home page.

The HTTP address for this home page is <http://moralforce.cc.gatech.edu/snakes/index.html>. Processing examples are shown in Figure 3.13.

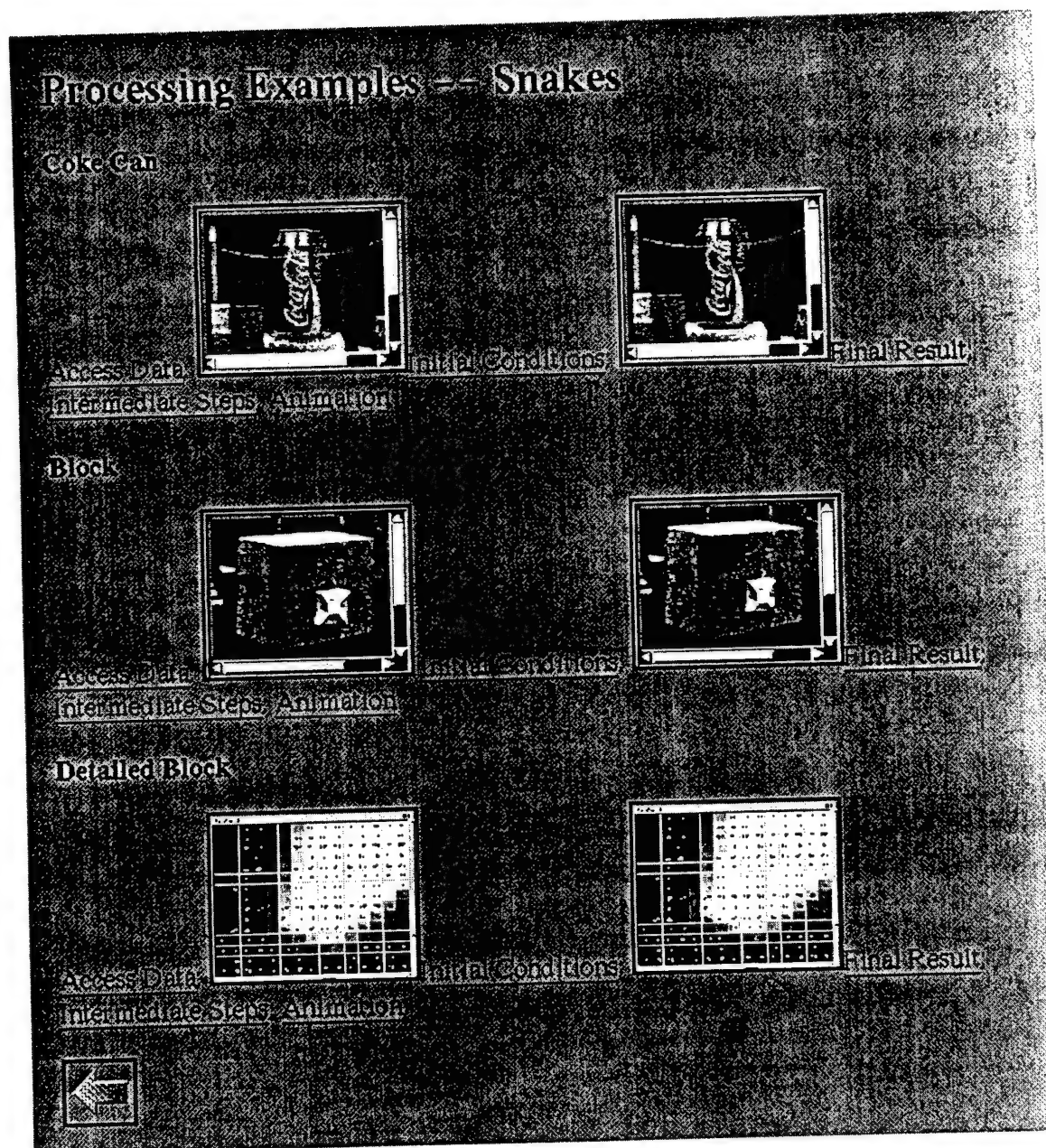


Figure 3.13. Processing examples.

The initial point positions for the snake are shown in Figure 3.14.

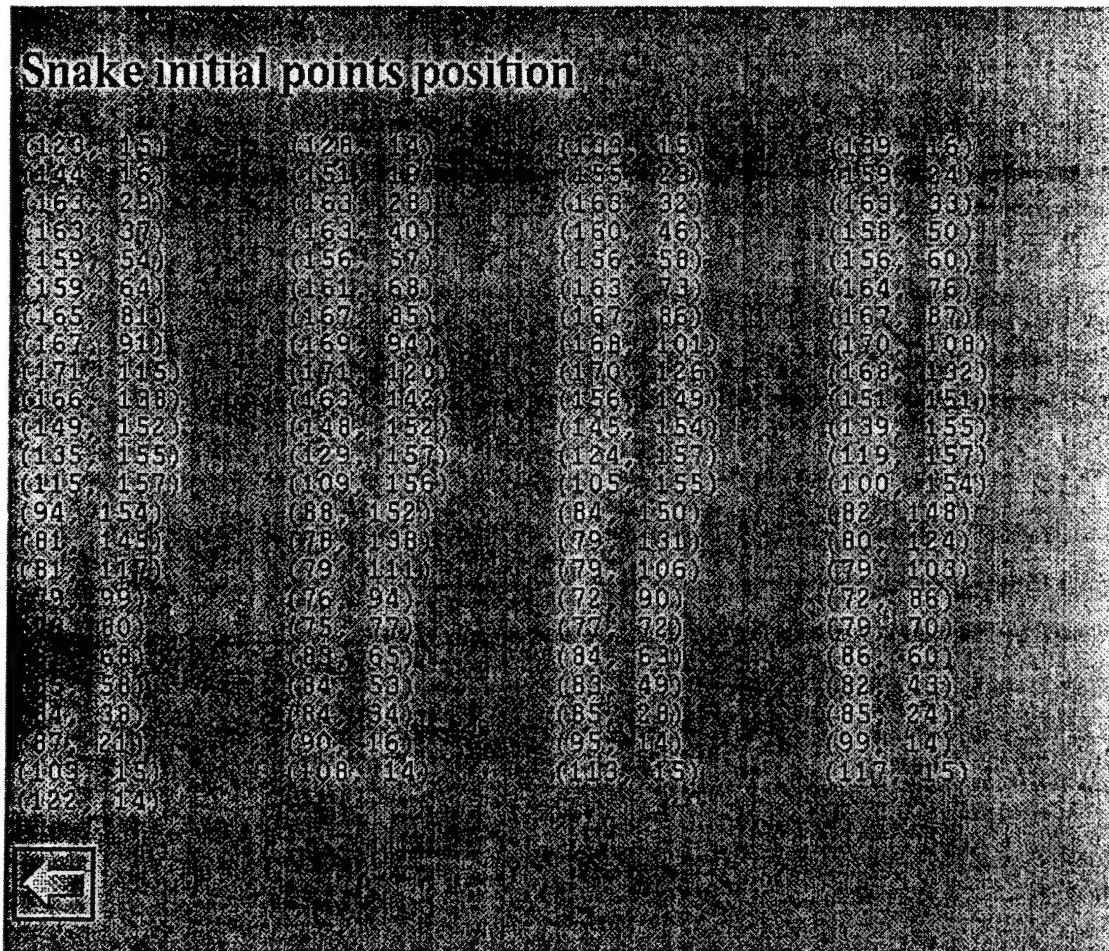


Figure 3.14. Initial point positions for the snake.

The initial position of the snake displayed as a graphical overlay on the image is shown in Figure 3.15.

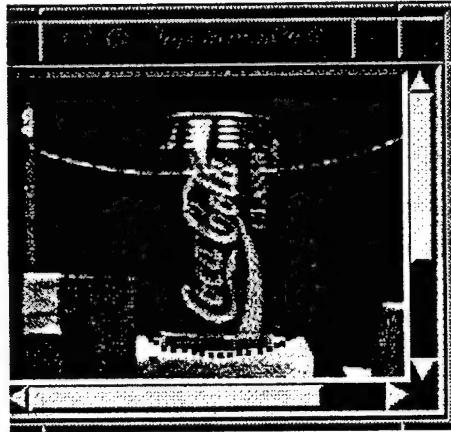


Figure 3.15. Initial snake position.

The final position of the snake displayed as a graphical overlay on the image is shown in Figure 3.16.

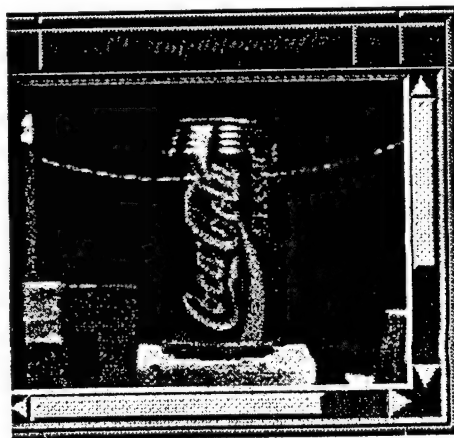


Figure 3.16. Final snake position.

The snake position is shown for some of the 25 iterations in Figure 3.17.

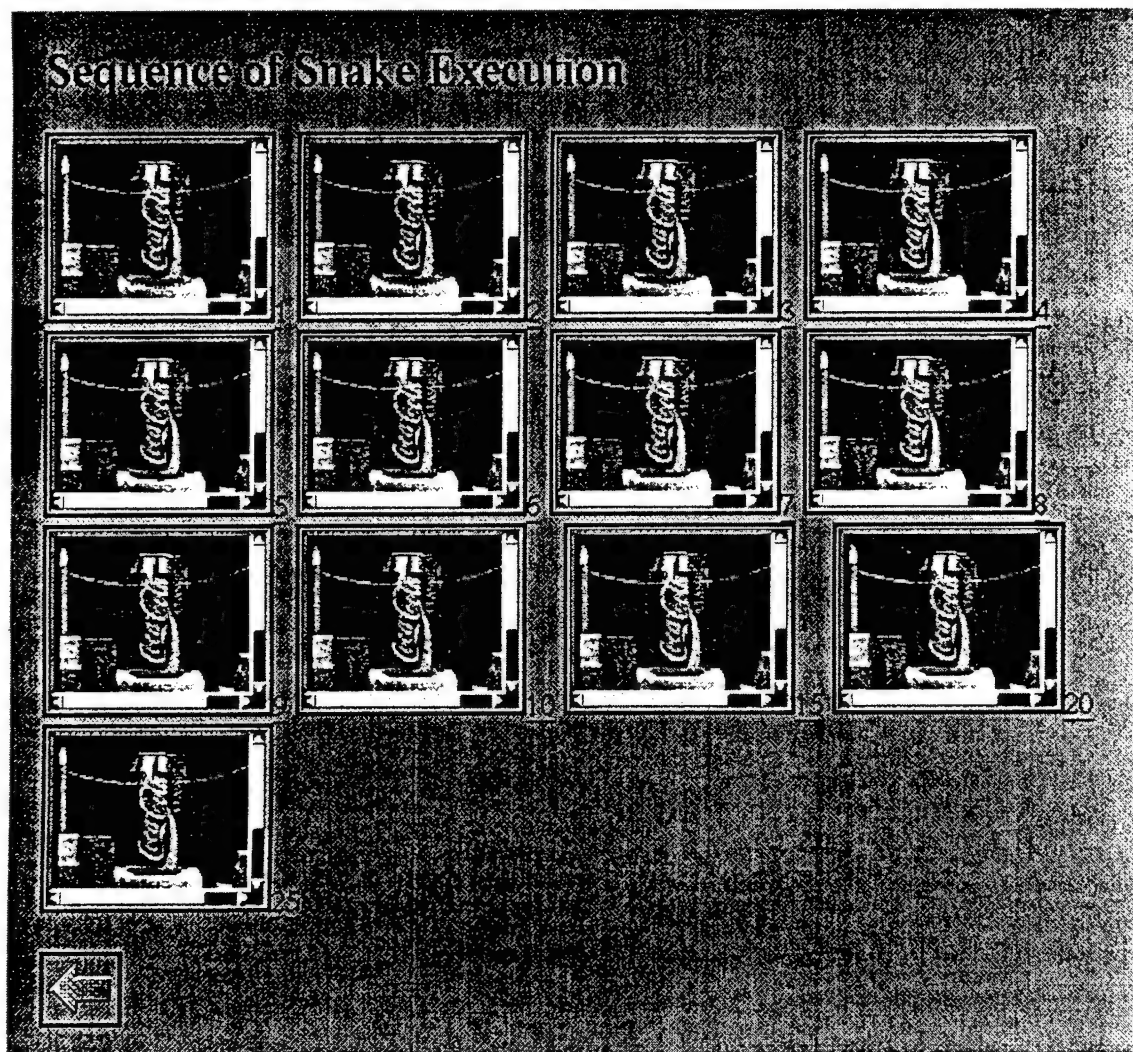


Figure 3.17. Snake position at different iterations.

A code fragment from the tutorial is shown in Figure 3.18.

```

001  // minEnergy: min energy, min position: min snake position
002  // minEnergy: min energy, min position: min snake position
003  // minEnergy: min energy, min position: min snake position
004  // minEnergy: min energy, min position: min snake position
005  // minEnergy: min energy, min position: min snake position
006  // minEnergy: min energy, min position: min snake position
007  // minEnergy: min energy, min position: min snake position
008  // minEnergy: min energy, min position: min snake position
009  // minEnergy: min energy, min position: min snake position
010  // minEnergy: min energy, min position: min snake position
011  // minEnergy: min energy, min position: min snake position
012  // minEnergy: min energy, min position: min snake position
013  // minEnergy: min energy, min position: min snake position
014  // minEnergy: min energy, min position: min snake position
015  // minEnergy: min energy, min position: min snake position
016  // minEnergy: min energy, min position: min snake position
017  // minEnergy: min energy, min position: min snake position
018  // minEnergy: min energy, min position: min snake position
019  // minEnergy: min energy, min position: min snake position
020  // minEnergy: min energy, min position: min snake position

```

Figure 3.18. Code fragment from the tutorial.

Clicking on the variable `nextNode` in the program fragment brings up the explanation shown in Figure 3.19.

`nextNode` contains the coordinates for the neighbor of the next node indexed by `i`. These coordinates are used only to compute the energy associated with the associated choice of snake positions.




Figure 3.19. Explanation associated with the variable `nextNode`.

The snakes tutorial home page is shown in Figure 3.20.

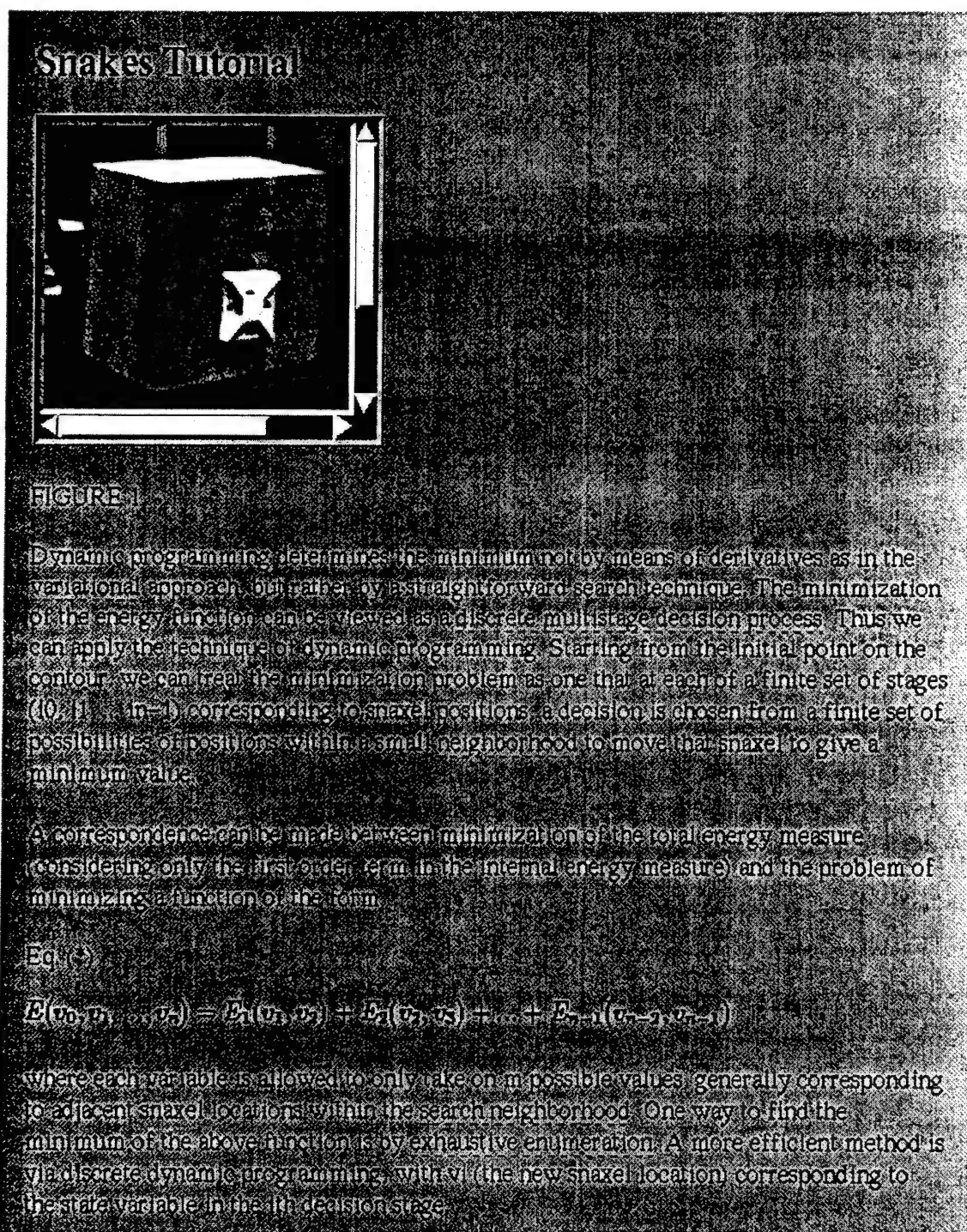


Figure 3.20. Snakes tutorial home page.

Clicking on the word neighborhood brings up the explanation shown in Figure 3.21.

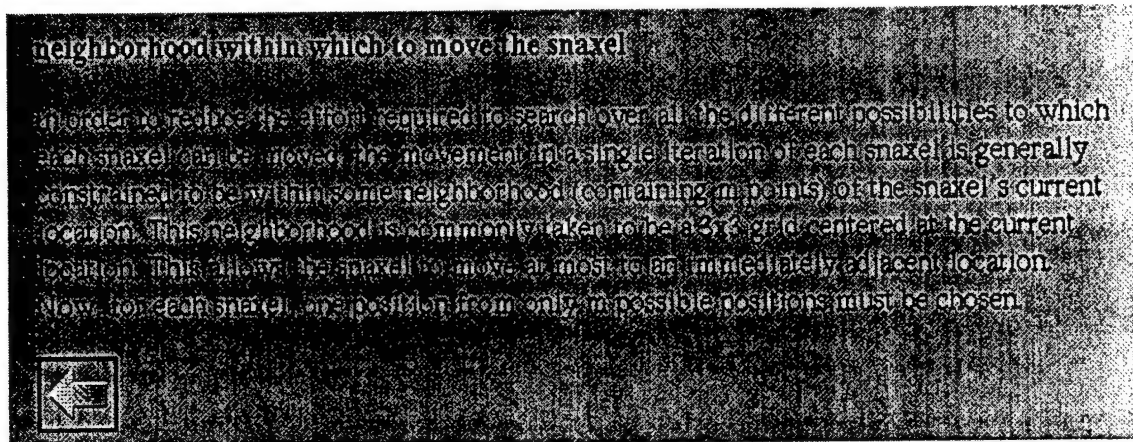


Figure 3.21. Explanation of neighborhood.

Chapter 4

Knowledge Weasel

Knowledge Weasel (KW) is a document annotation system written in Tcl/Tk. KW has the ability to annotate text documents, as well as images and simple drawings. Hyper-text buttons within text documents or graphical documents allow the user to jump to other documents. Hyper-text buttons may be used to display additional graphics, or overlays, on existing graphical documents.

KW also allows the creating of "reactive annotations." The user may select a portion of a document and then select his "reaction" from a predefined list. Later, the author of the document may retrieve the users' reactions.

Annotation data is stored within a database, such as oracle. KW is integrated with this database, allowing annotations to be limited by specific search criteria. An accompanying database browser is used to make the search, and results are then passed to KW and the desired documents and annotations are displayed.

KW is written entirely in Tcl/Tk using XF. The pixmap, tree, and photo widget extensions are compiled into the Tcl/Tk wish. The database browser that is used in conjunction with KW is also written in Tcl/Tk with the TclX and OraTcl extensions. The two applications communicate through the use of the Tk "send" command, which allows Tk applications to send commands to one another.

4.1 Components

KW consists of four main components. These are the file viewer, the file creator, the annotation creator, and the annotation tree viewer.

The file viewer is used to view existing KW files. Annotations are shown as buttons inserted into the text, or overlaid on graphical documents. Buttons are delimited by <*>. Buttons may appear in a different color and font than the rest of the document, as specified by its default behavior or by the database browser. Whenever a user opens a KW file, it is displayed using the file viewer. The text file viewer is used for displaying text documents, while the image file viewer is used for displaying graphical documents.

The file creator is used for creating new KW files. This component has two incarnations: the text file creator and the graphical file creator. New documents may be typed into the text file creator. In addition, the text file creator has menu options to load and edit existing text files, or to insert existing text files into the document.

The graphical file creator is much more complex than the text file creator. The graphical file creator allows the user to draw new graphical diagrams. The graphical file creator supports a number of graphical primitives. These primitives include lines with or without arrowheads, multi-segmented

lines with or without arrowheads, outlined rectangles, filled rectangles, outlined ovals, filled ovals, filled polygons, outlined polygons, bitmaps or pixmaps, and text. In addition, primitives may be moved or erased, and color and line thickness can be selected. Finally, the user has the ability to insert "sleep times" of various duration. This allows graphical files to contain rudimentary animation.

The annotation creator is used when creating new annotations. After a region of a document has been selected in the file viewer, and the "Annotate" menu option has been selected, the annotation creator will appear. The annotation creator itself has two components: the annotation control panel and the annotation content creator. The annotation control panel is where the user sets or views the default characteristics of the annotation, while the annotation content creator is where the actual contents of the annotation is created.

The annotation control panel has a number of characteristics the user can set. The Button Text is the actual text that will be seen in the button between the "<*" and "*>" delimiters. The Button Text may not contain any white space. The Button Font is the default font for the text of the button. The font of the Button Text field reflects changes to the Button Font. The Active Color is the default color of the button when the pointer is positioned over it. The Inactive Color is the default color of the button when the pointer is not positioned over it. The Annotation Type determines whether the button will bring up another text file, a new graphical document, or an overlay to an existing graphical document. The Button Location field indicates in which file the new annotation button is located. The Primary Target indicates the name of the new file being created, or, in the case of an overlay, the name of the file being overlayed. For an overlay, the Secondary Target field shows the name of the new file being created.

The annotation content tool has two primary incarnations. If a text annotation is being created, the text incarnation will be displayed. This is similar to the text file creator used for creating new text documents. If a canvas annotation is being created, the graphical incarnation will be displayed. This is similar to the graphical file creator used for creating new graphical documents. If an overlay annotation is being created, the graphical incarnation will be displayed, but the user first selects an existing graphical file to load into it. The new overlay file will then exist "on top of" the previous graphical file. The original graphical file is not modified in any way.

The annotation tree viewer displays the relationships between KW graphical files. Only files that have been loaded by KW are displayed in the tree viewer. The tree viewer may be used to load files by clicking on file names that appear in the tree viewer.

4.2 Example of Use

Following is a simple example of how one would use Knowledge Weasel to create and annotate documents.

When KW is first executed, the main control panel is displayed (Figure 4.1).

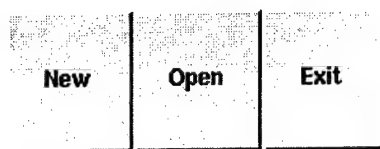


Figure 4.1. Main control panel.

The user may select "New" to create a new KW file, "Open" to open an existing KW file, or "Exit" to exit KW. Selecting the "New" button will make the file creator component. If the file creator appears in its graphical incarnation, change it to the text incarnation by selecting the option "Text" from the pull-down menu under the word "Canvas."

Once the file creator is in the text incarnation, new text may be entered in the text area. Also, existing text files may be loaded into the text area or inserted into the text area by using the File menu (Figure 4.2).

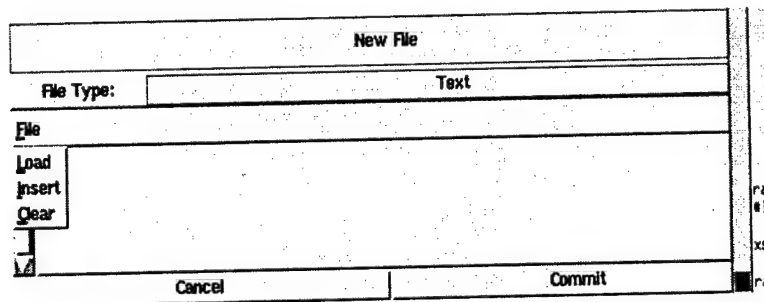


Figure 4.2. File menu.

Once the text has been entered, the file may be committed to KW by pressing the "Commit" button. This brings up the "Commit File" window (Figure 4.3).

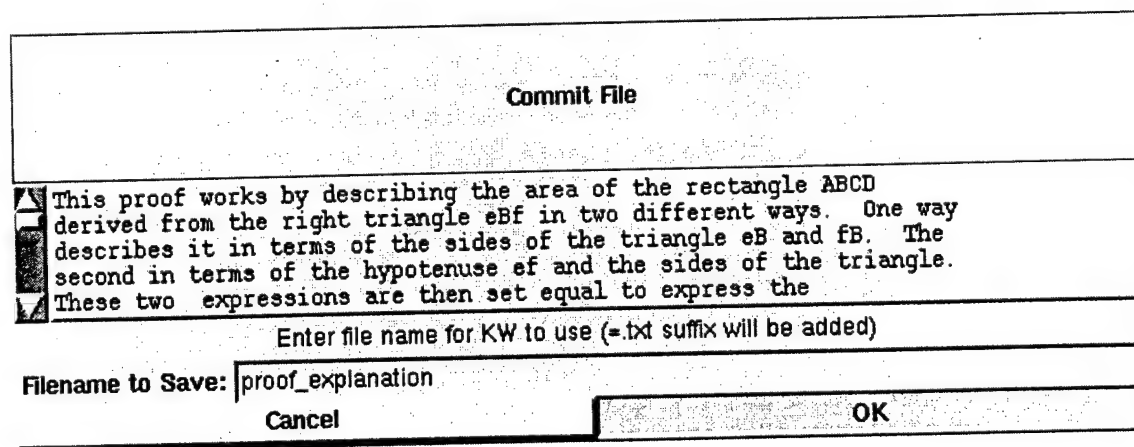


Figure 4.3. Commit File window.

This window shows the contents of the file and asks for a name under which to save the new KW file. Since this is a new file, KW will automatically add a “=” to the name entered. Since this is also a text file, a “.txt” extension will be added as well. At this time it is possible to hit “Cancel” and return to the file creator. If everything is as desired, press “OK.” This will save the text file in the predefined KW storage directory.

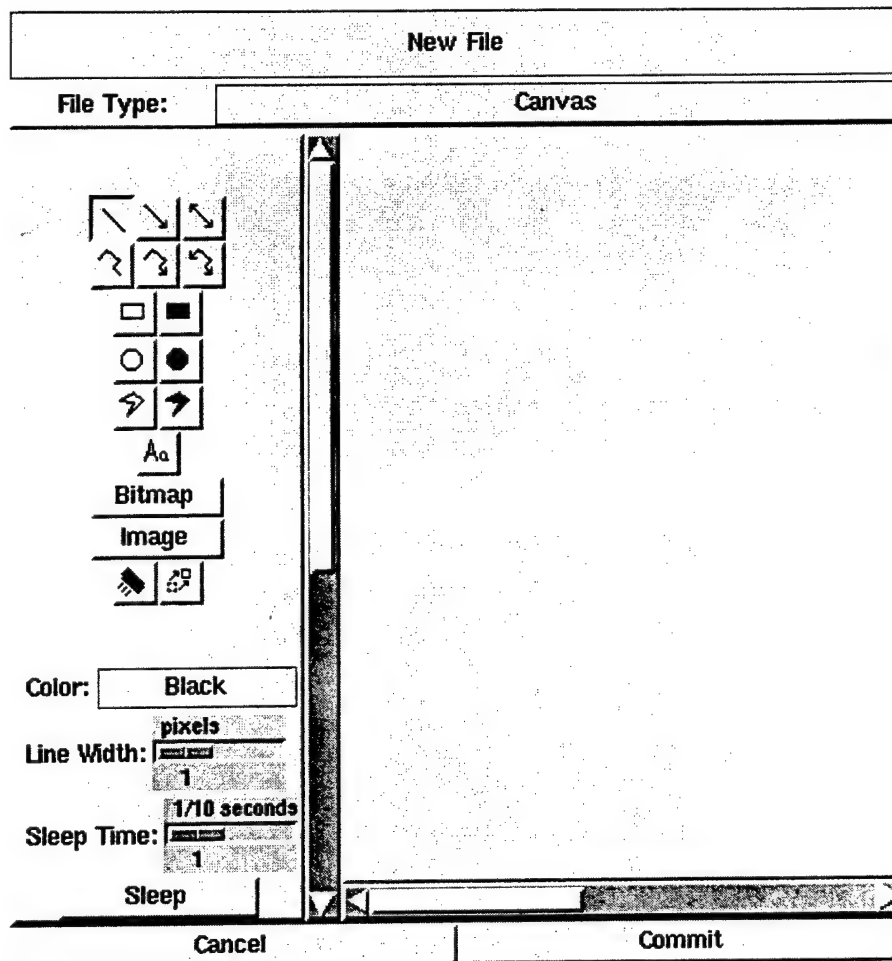


Figure 4.4. Canvas file type.

Next, we will create a new “canvas” file. The file creator is once again brought up by the “New” button from the main control panel. This time, select the “Canvas” file type (Figure 4.4). In this example, a large bitmap on the canvas will be displayed. First, select “Bitmap” from the panel on the left side of the file creator. Next, on the right side canvas of the file creator, select where the upper left corner of the bitmap is to be placed. This location will be indicated on the canvas and a file selector will appear. Using this file selector, find the desired bitmap or pixmap. The bitmap will then be copied into the KW storage directory. If a file with the same name is already present in the storage directory, the contents of the files will be compared, and if the files are the same, the existing file will be used. If the two files are not the same, the name of the new file will be changed before being copied into the storage directory.

The bitmap is then displayed in the canvas (Figure 4.5)

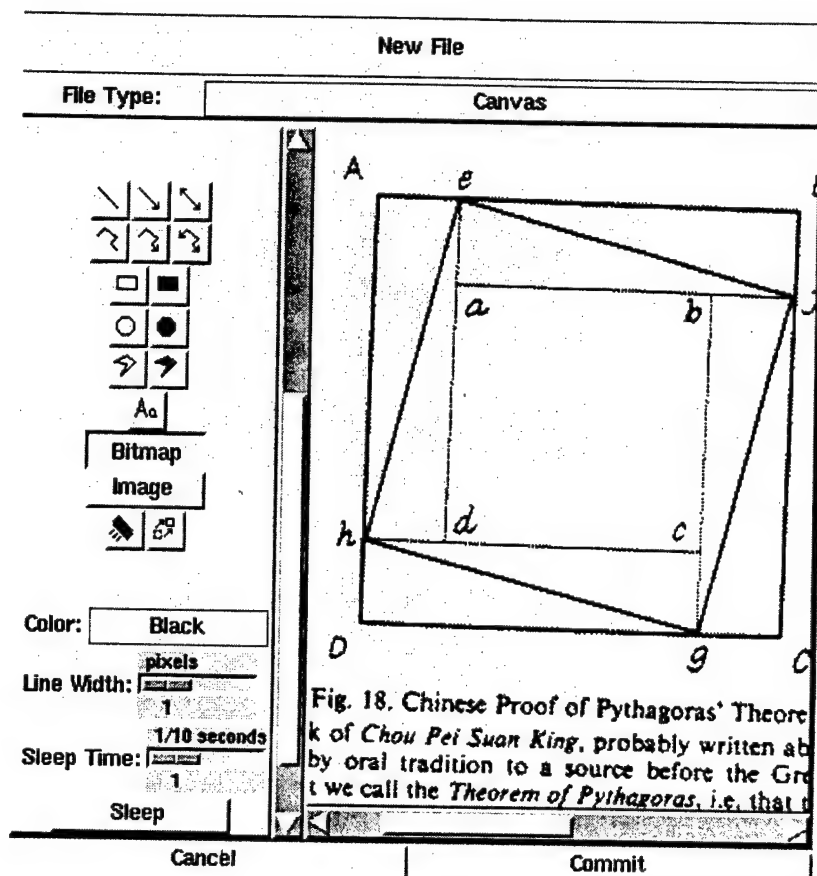


Figure 4.5. Bitmap display

Committing a canvas file brings up the “Commit File” window as with text files. For canvas files, this window shows the actual commands necessary to recreate the canvas. Again, since this is a new file, “=” will be added to the file name entered. Since this is a canvas file, “.cvs” also will be added (Figure 4.6).

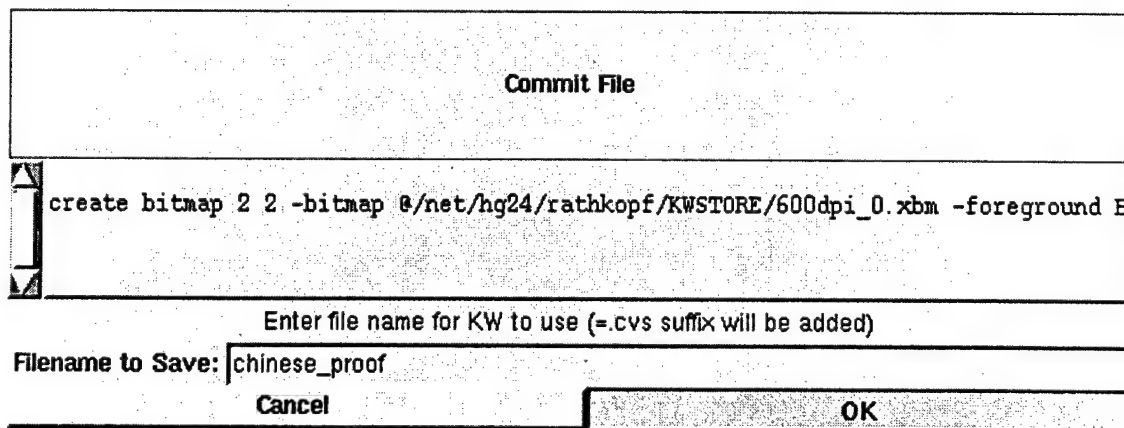


Figure 4.6. Saving a canvas file.

To open an existing KW file, select "Open" from the main control panel. This will bring up a file selector limited to the KW storage directory. The selection pattern can be used to limit the files displayed. For instance, "*" will list all original KW files, while "*#*" will list all KW annotation files, and "*[=#]*)" will list all original and annotation files (Figure 4.7).

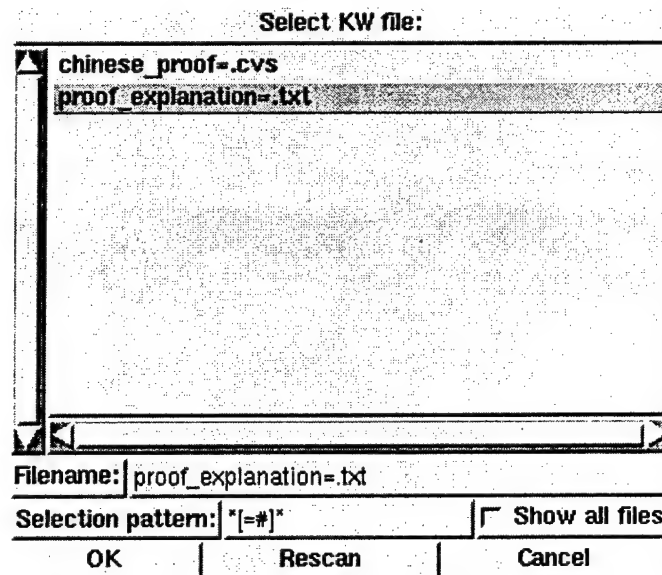


Figure 4.7. File selector.

Once a file has been selected, it will be displayed in the appropriate file viewer. If any hyper-text buttons exist in the specified file, they will be displayed. To annotate a text file, select the portion of text that you wish to annotate and select "Annotate" from the "Annotate" menu (Figure 4.8).

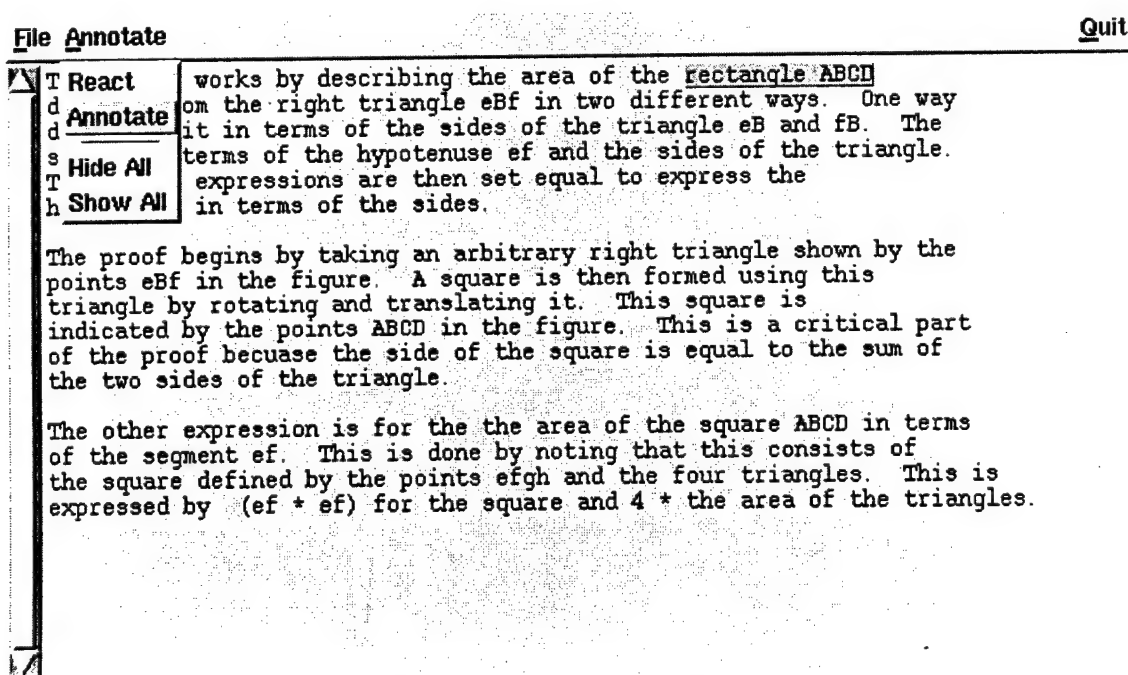


Figure 4.8. Annotate menu.

This will bring up the annotation attribute window and the annotation file window. The annotation attribute menu is used to specify the actual text of the button, the active and inactive colors of the button, the font of the button, the type of annotation, the role of the annotation, and keywords associated with the annotation (Figure 4.9).

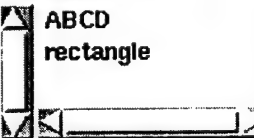
Button Text:			
Active Color:	Red	Inactive Color:	Blue
Font:	Helvetica	14pt	Bold Straight
Annotation Type:	Overlay	Annotation Role:	Illustration
Keywords:			
Button Location:	Primary Target:	Secondary Target:	
proof_explanation=.txt	chinese_proof=.cvs	proof_explanation#_0.ovl	
Coordinates:	1.47:1.61	Coordinates:	
Author Name:	Ted Rathkopf	Annotation Date:	08-Dec-94 14:36:35

Figure 4.9. Annotation attribute menu.

Changing the “Annotation Type” will change the form of the annotation file window. If “Overlay” is selected, a file selection window will appear to select the canvas or overlay for which you wish to create an overlay.

The annotation file window is similar to the file creator window. The primary difference is that when an overlay annotation is being made, the canvas does not start out blank, but rather contains the file being overlaid. Graphical items can then be drawn on top of the previous file (Figure 4.10).

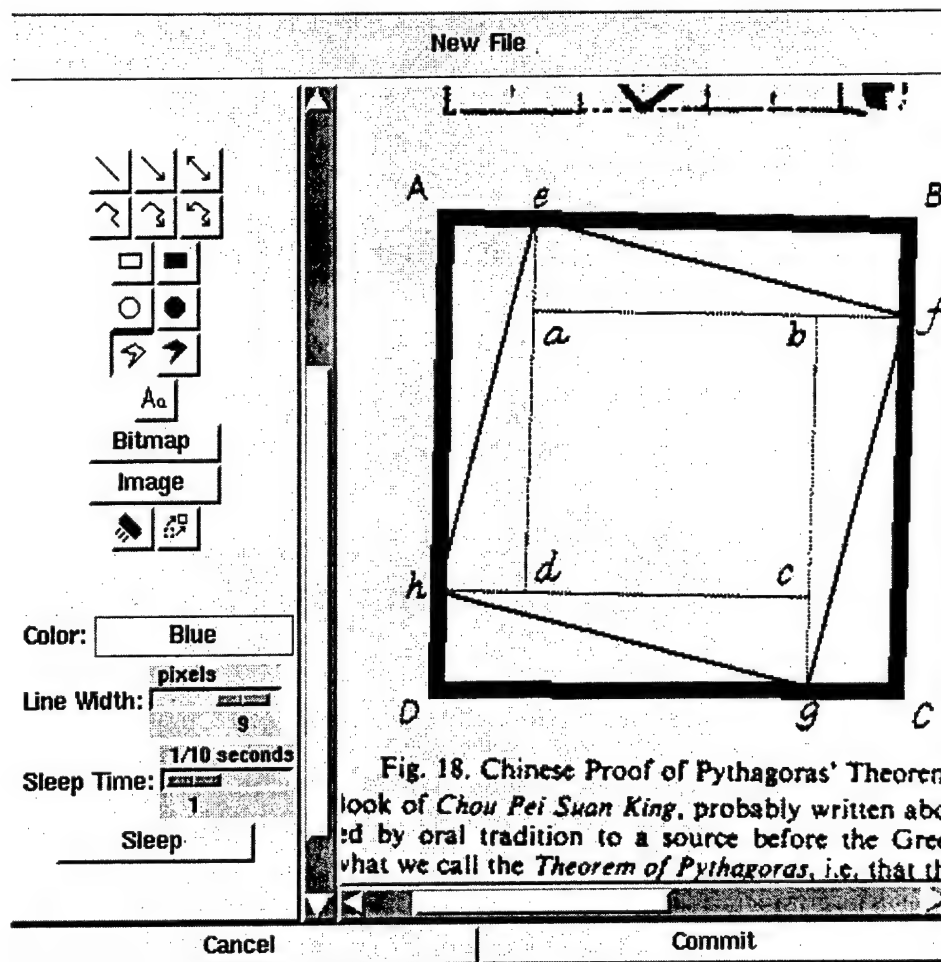


Figure 4.10. Annotation file window.

These additions are stored in the new overlay file and the original canvas or overlay is not modified in any way.

It is possible to create separate and distinct overlays on the same canvas file. In the example, an overlay for triangle eBf, which is separate from the overlay for rectangle ABCD, can be created (Figures 4.11 and 4.12).

Button Text:			
Active Color:		Red	Inactive Color:
		Blue	
Font:	Helvetica	14pt	Bold
		Straight	
Annotation Type:		Overlay	Annotation Role:
		Illustration	
Keywords:		<div> <div>eBf</div> <div>triangle</div> </div>	
Button Location:		Primary Target:	Secondary Target:
proof_explanation=.txt		chinese_proof=.cvs	proof_explanation#_1.ovl
Coordinates:	2.23:2.35	Coordinates:	Coordinates:
Author Name:		Ted Rathkopf	Annotation Date:
		08-Dec-94 14:41:10	

Figure 4.11. Annotation attributes for triangle eBf.

File	Annotate	Quit
<p>This proof works by describing the area of the rectangle ABCD<*> derived from the right triangle eBf<*> in two different ways. One way describes it in terms of the sides of the triangle eB and fB. The second in terms of the hypotenuse ef and the sides of the triangle. These two expressions are then set equal to express the hypotenuse in terms of the sides.</p> <p>The proof begins by taking an arbitrary right triangle shown by the points eBf in the figure. A square is then formed using this triangle by rotating and translating it. This square is indicated by the points ABCD in the figure. This is a critical part of the proof because the side of the square is equal to the sum of the two sides of the triangle.</p> <p>The other expression is for the the area of the square ABCD in terms of the segment ef. This is done by noting that this consists of the square defined by the points efgh and the four triangles. This is expressed by (ef * ef) for the square and 4 * the area of the triangles.</p>		

Figure 4.12. Overlay for triangle eBf.

If the overlay file for triangle eBf is opened, a graphical file viewer window will appear with the original canvas displayed with triangle eBf shown (Figure 4.13).

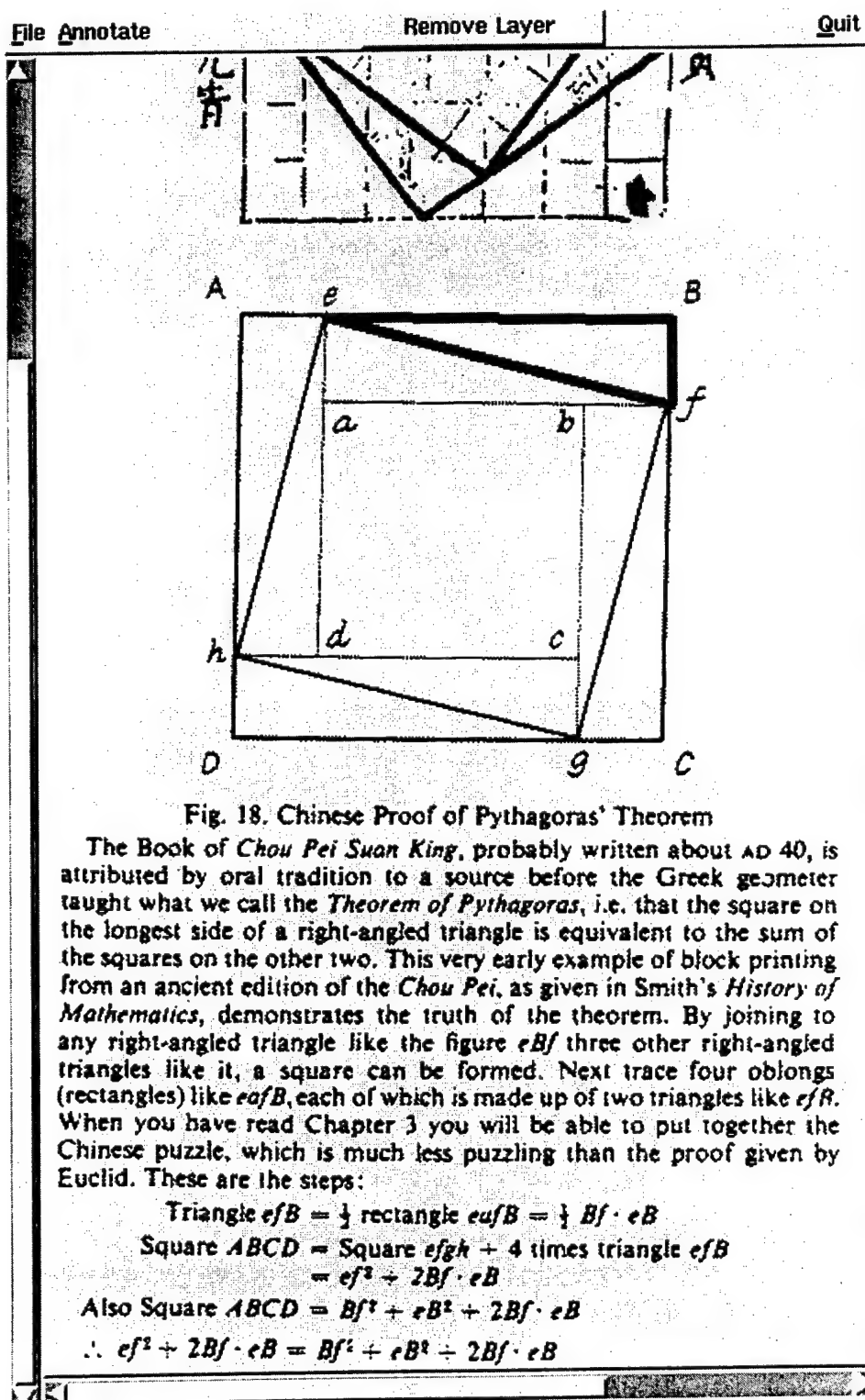


Figure 4.13. Graphical viewer window for triangle eBf

The "Remove Layer" button at the top of the graphical file viewer may be used to "peel off" overlays, one at a time, down to the starting canvas.

The annotation tree viewer shows the relationship of canvas files and overlays. Once an overlay or canvas has been loaded it will be shown in the tree viewer. For example, the original canvas has two separate and distinct overlay files (Figure 4.14).

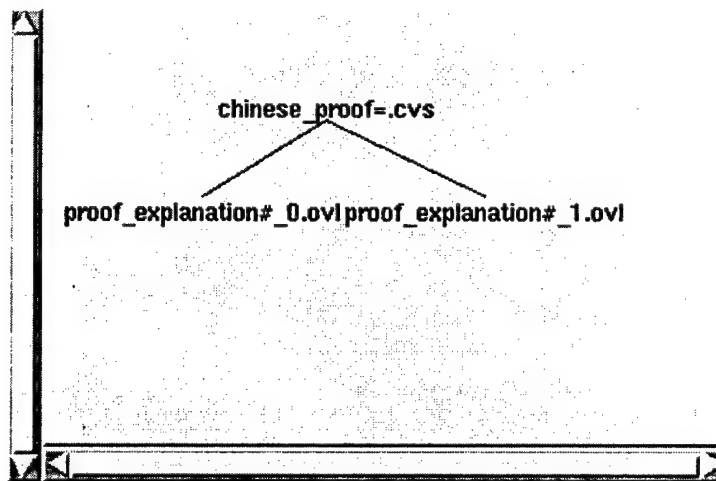


Figure 4.14. Annotation tree viewer

The files currently being displayed are shown in blue while files not being displayed are shown in black. Clicking on the name of a file will cause that file to be displayed.

Annotation buttons may exist in graphical files as well as text files. To create an annotation on a graphical file, select a region of the canvas to which the annotation pertains. Next select "Annotate" from the "Annotate" menu. Since more than one "layer" may be displayed, it is

necessary to select the layer to which the button belongs. Therefore, the "Annotate" entry in the menu requires a further selection of the layer to be annotated (Figure 4.15).

File Annotate
Remove Layer
Quit

React

Annotate
chinese_proof=.cvs
proof_explanation#_1.ovl

Fig. 18. Chinese Proof of Pythagoras' Theorem

The Book of *Chou Pei Suan King*, probably written about AD 40, is attributed by oral tradition to a source before the Greek geometer taught what we call the *Theorem of Pythagoras*, i.e. that the square on the longest side of a right-angled triangle is equivalent to the sum of the squares on the other two. This very early example of block printing from an ancient edition of the *Chou Pei*, as given in Smith's *History of Mathematics*, demonstrates the truth of the theorem. By joining to any right-angled triangle like the figure *eBf* three other right-angled triangles like it, a square can be formed. Next trace four oblongs (rectangles) like *efB*, each of which is made up of two triangles like *eBf*. When you have read Chapter 3 you will be able to put together the Chinese puzzle, which is much less puzzling than the proof given by Euclid. These are the steps:

Triangle *efB* = $\frac{1}{2}$ rectangle *efB* = $\frac{1}{2}$ *Bf* · *eB*

Square *ABCD* = Square *efgh* + 4 times triangle *efB*

= *ef*² + 2*Bf* · *eB*

Also Square *ABCD* = *Bf*² + *eB*² + 2*Bf* · *eB*

∴ *ef*² + 2*Bf* · *eB* = *Bf*² + *eB*² + 2*Bf* · *eB*

∴ *ef*² = *Bf*² + *eB*²

50

Figure 4.15. Selecting a "layer".

Creation of the annotation file proceeds as before. Once the annotation file has been selected, it is necessary to choose the placement of the new annotation button in the graphical file. As in the example, in the prior figures, the button is placed right above the word "rectangle" in the proof at the bottom of the page. The "Button Location" for this button is the bottom layer canvas file.

File Annotate
Remove Layer
Quit

Fig. 18. Chinese Proof of Pythagoras' Theorem

The Book of *Chou Pei Suan King*, probably written about AD 40, is attributed by oral tradition to a source before the Greek geometer taught what we call the *Theorem of Pythagoras*, i.e. that the square on the longest side of a right-angled triangle is equivalent to the sum of the squares on the other two. This very early example of block printing from an ancient edition of the *Chou Pei*, as given in Smith's *History of Mathematics*, demonstrates the truth of the theorem. By joining to any right-angled triangle like the figure *efB* three other right-angled triangles like it, a square can be formed. Next trace four oblongs (rectangles) like *efB*, each of which is made up of two triangles like *efB*. When you have read Chapter 3 you will be able to put together the Chinese puzzle, which is much less puzzling than the proof given by Euclid. These are the steps: <+>

Triangle *efB* = $\frac{1}{2}$ rectangle *efB* = $\frac{1}{2} Bf \cdot eB$

Square *ABCD* = Square *efgh* + 4 times triangle *efB*

$= ef^2 + 2Bf \cdot eB$

Also Square *ABCD* = $Bf^2 + eB^2 + 2Bf \cdot eB$

$\therefore ef^2 + 2Bf \cdot eB = Bf^2 + eB^2 + 2Bf \cdot eB$

$\therefore ef^2 = Bf^2 + eB^2$

50

Figure 4.16. Display of multiple overlays.

The annotation is an overlay on the overlay containing triangle eBf. This button will be present in the display of any overlay containing this original canvas, and selecting this button will result in the display of the new overlay and the overlay of triangle eBf (Figure 4.16). The relationship of the overlays can be seen in Figure 4.17.

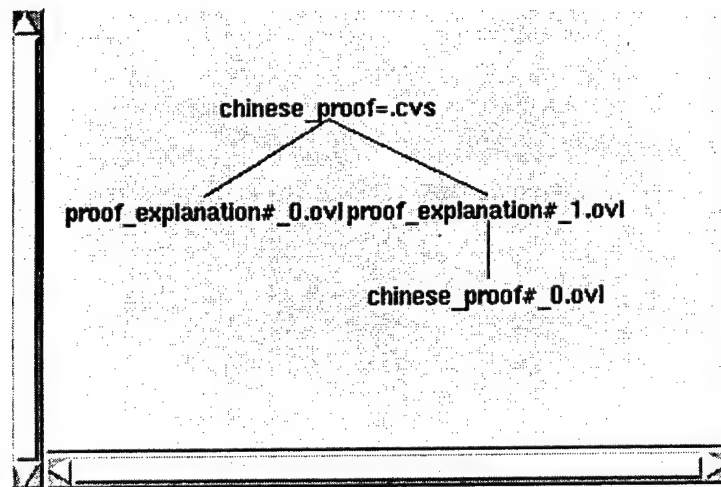


Figure 4.17. Tree view of multiple overlays.

4.3 The Database Browser

The Database Browser provides a graphical interface to the annotation database. Database queries are displayed in the main database browser window. Filters can be entered in the database browser window to limit the database query results (Figure 4.18).

File	History	Results	Options	Help
Query Filter				
Execute				
Query Results				
TARGET1_FILE	TARGET2_FILE	KEYWORD		
-----	-----	-----		
chinese_proof=.cvs	proof_explanation#_0	ABCD rectangle		
chinese_proof=.cvs	proof_explanation#_1	eBf triangle		
proof_explanation#_1	chinese_proof#_0.ovl			

SQL finished, 3 rows returned, 3 rows affected

Figure 4.18. Database Browser.

The Attribute Selector is used to select which database fields are displayed in the main window (Figure 4.19).

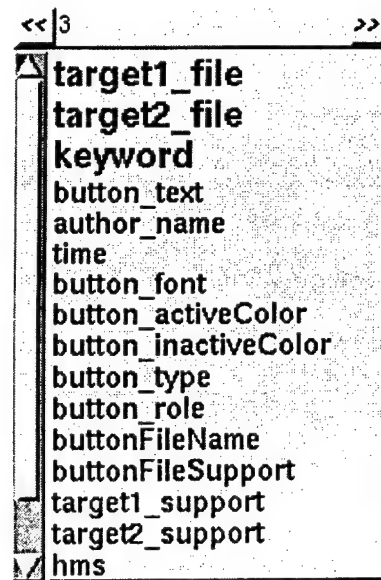


Figure 4.19. Attribute Selector.

The Attribute Selector allows the number of displayed fields to be chosen, as well as the order of the fields to be displayed. All annotation attribute fields are available for selection. The Query Keypad can be used to simplify the construction of query filters (Figure 4.20).

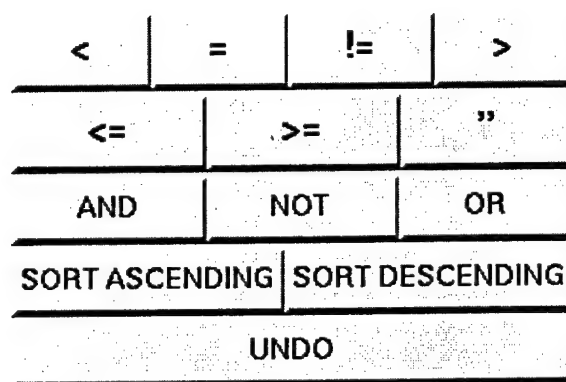


Figure 4.20. Query Keypad.

Finally, the Mapper can be used to map the results of a database query back to KW (Figure 4.21).

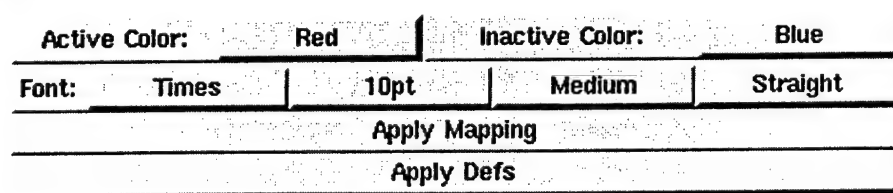


Figure 4.21. Mapper.

The visible characteristics of the buttons, such as color and font, can be changed in the mapper to override the annotation defaults.

For example, we can construct the query “button_role = 'Illustration'” for the database as it would appear after the above examples. If only the “Buttonfilename” and “Button_Role” attributes are selected for viewing, the results of the search will be as in Figure 4.22.

FileHistoryResultsOptionsHelp

Query Filter

button_role = 'Illustration'

Execute

Query Results

BUTTONFILENAME	BUTTON_ROLE
proof_explanation=.t	Illustration
proof_explanation=.t	Illustration

SQL finished, 2 rows returned, 2 rows affected

Figure 4.22. Example query with two attributes selected for viewing.

Selecting font and colors, as in Figure 4.23, and applying the mapping, will cause KW to display the file as in Figure 4.24.

Active Color:		White		Inactive Color:		Black	
Font:	Times	18pt	Bold	Italic			
Apply Mapping							
Apply Defs							

Figure 4.23. Selecting font and colors for the mapping.

File Annotate
Quit

This proof works by describing the area of the rectangle ABCD <*>
 derived from the right triangle eBf in two different ways. One way
 describes it in terms of the sides of the triangle eB and fB. The
 second in terms of the hypotenuse ef and the sides of the triangle.
 These two expressions are then set equal to express the
 hypotenuse in terms of the sides.

The proof begins by taking an arbitrary right triangle shown by the
 points eBf in the figure. A square is then formed using this
 triangle by rotating and translating it. This square is
 indicated by the points ABCD in the figure. This is a critical part
 of the proof because the side of the square is equal to the sum of
 the two sides of the triangle.

The other expression is for the area of the square ABCD in terms
 of the segment ef. This is done by noting that this consists of
 the square defined by the points efgh and the four triangles. This is
 expressed by $(ef * ef)$ for the square and $4 * \text{the area of the triangles}$.

Figure 4.24. Example query mapping.

Chapter 5

Gravity

Measurements of the direction of gravity can be fused with visual input to provide more accurate motion estimates, and to draw conclusions about environmental structure. In this chapter this concept is illustrated through a technique for extracting structure and motion from line correspondences and a priori knowledge of the direction of gravity.

There exists neurological evidence that the perception of balance is important to the human visual system [6]. The vestibular system of the brain and inner ear gives us our sense of balance. The vestibular organs signal the brain about the direction of gravity and the acceleration produced during head movements. These signals are used to produce reflexes that maintain head position as well as control eye movements. This is important for keeping images fixed on the retina, but also may be used by the visual cortex for higher level processes.

Psychophysical studies have also shown that the perception of balance is important to the human visual system. Two experiments described by Irvin Rock [17] suggest the importance of gravity in perceiving the visual orientation of objects in the environment. In the first experiment, an observer stands in a small dark room that is revolving in a circular path. The centrifugal force together with gravity produces a resultant force in an oblique direction. When the observer is asked to place a luminous rod in a vertical position, it is placed in the direction of the resultant force. In the second experiment, an observer is placed in a tilted position in a dark room. Once again the observer is asked to place a luminous rod in a vertical position. The observers have no trouble doing this despite the fact that the image of the rod on the retina is tilted. These experiments suggest that gravity is used by the visual system for determining environmental orientation.

In this chapter, an algorithm for computing structure and motion from line correspondences is presented. Line features are preferred over point features, since matching lines do not require the determination of exact 2-D displacements. In order to constrain the possible motion configurations, one assumes that the 3-D direction of gravity relative to each image frame is known. The algorithm presented is linear and can incorporate an arbitrary number of images, resulting in robust estimation of the parameters of motion.

The next section discusses the feasibility of obtaining a 3-D gravity vector. Section 5.2 introduces the notation used throughout the chapter. Section 5.3 shows how to derive structure and motion from a sequence of 2-D images and corresponding 3-D gravity vectors. The performance of this algorithm in the presence of noise is analyzed in Section 5.4 by applying the algorithm to synthetic image sequences. Finally, results obtained from a real image sequence are presented in Section 5.5.

5.1 Recovery of Gravity

Gravity can be recovered in several different ways. The most obvious method for recovering gravity is to use a gravitational sensor such as an inclinometer. Inclinometers are extremely cheap

in comparison to the costs of the cameras, digitizers, and computers necessary for building vision systems. One drawback to using an inclinometer is that this sensor measures the current direction of acceleration. Therefore, the camera must undergo constant linear acceleration in order for this direction to remain constant. This assumption is valid for small accelerations, such as those experienced by a camera mounted on an indoor robot, but not valid for larger accelerations, such as those undergone by a camera mounted on an outdoor vehicle.

Another method of recovering the gravitational direction is to use lines parallel to gravity. The projections of parallel lines can be intersected, and the vanishing point used to recover the 3-D direction of gravity. Man-made objects, such as those typically found in indoor scenes, usually contain many parallel lines. Initially these lines can be grouped together automatically based upon a common vanishing point, or they can be extracted interactively in a system such as that presented in Chapter 6. Once a set of parallel lines has been extracted, it can be tracked throughout the sequence, and the direction of gravity can be recovered.

5.2 Notation

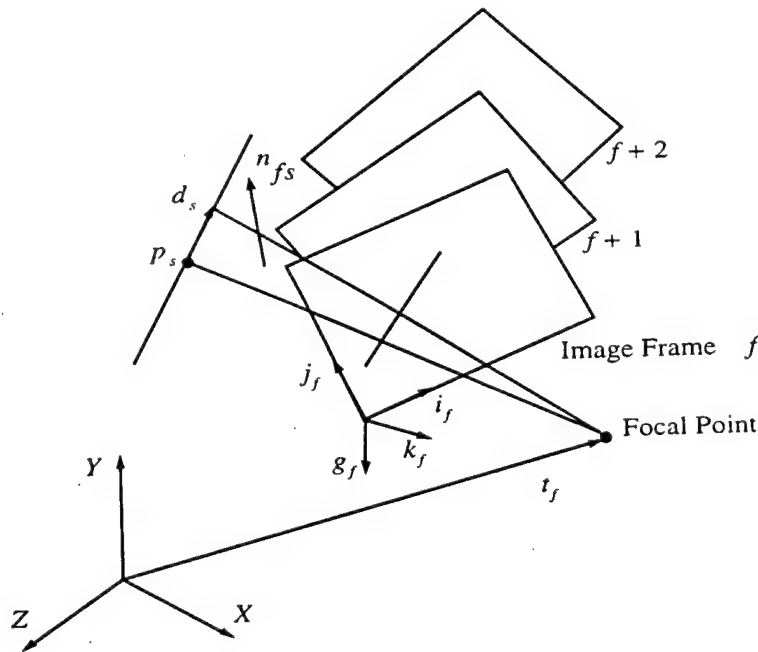


Figure 5.1. Gravity coordinate system.

The notation used throughout this chapter is shown in Figure 5.1. The orientation of an image frame at time f is delineated by unit vectors i_f , j_f , and k_f . The position of the image frame is specified by t_f . A 3-D environmental line is represented by a unit vector d_s specifying the line direction, and a point on the line p_s . Line (d_s, p_s) is projected onto image frame f . The projection of this line and the focal point of the camera define a plane. The normal to this plane is n_{fs} . The direction of gravity will be referred to as g_f . n_{fs} and g_f are expressed in the coordinate system of image frame f . All other parameters are specified relative to the world coordinate system.

Section 5.3.2 is concerned with solving for the line orientations d_s , as well as the parameters of rotation i_f , j_f , and k_f . Section 5.3.3 shows how these initial quantities can be used to fix the relative positions of the lines within the world coordinate system by solving for a point p_s on each line.

5.3 Structure and Motion Estimation

This section presents a method of solving for the structure and motion parameters given a set of lines, line correspondences, and 3-D gravity vectors. Only the line positions and orientations are used, since line endpoints are unstable. Line correspondences are used to solve for the parameters of motion with the following method:

1. Determine which lines are perpendicular to the known direction of gravity. These are the lines that will be used to solve for the parameters of motion. An algorithm for determining horizontal lines is given in Section 5.3.1.
2. Solve for the rotation matrix which aligns the Y-axis of each image frame with the image frame's 3-D gravity vector. This is done in Section 5.3.2.
3. Once the image frame has been aligned with gravity, the single unknown parameter of rotation is simply the amount of rotation about the gravity axis. The relationship between the 3-D line orientations and the amount of rotation can be expressed as a set of linear equations. These equations can be found in Section 5.3.2.
4. The final step in the line reconstruction is to solve for line positions relative to some world coordinate system. This is done in Section 5.3.3.

5.3.1 Horizontal Lines

The linear algorithm presented in the following section uses lines perpendicular to the direction of gravity to determine the angle of rotation about the gravitational axis. In this section, a rigidity relationship between lines is exploited in order to extract a subset of horizontal lines.

If a line is perpendicular to the direction of gravity, then we can solve for its 3D orientation with respect to the current image frame.

$$d_{fs} = \frac{n_{fs} \times g_f}{\|n_{fs} \times g_f\|} \quad (5.1)$$

The angle between two horizontal lines computed from Equation 5.1 will be constant for all images in a sequence. This will not be true for lines that are not perpendicular to the direction of gravity, since the 3-D lines computed using Equation 5.1 do not correspond to the true environmental structure. Therefore, the horizontal lines can be identified by calculating the sample variance of these angles over an image sequence. In the absence of noise, the variance will be zero for horizontal lines.

$$v = \frac{1}{n} \sum_{f=1}^n (\gamma_f - \bar{\gamma})^2 = 0 \quad (5.2)$$

In Equation 5.2, n is the number of image frames, γ_f is the angle between two candidate horizontal lines in image frame f , and $\bar{\gamma}$ is the sample mean.

Equation 5.2 is true for pairs of horizontal lines. However, parallel lines that lie close together in the image plane also will have very small variances. Therefore, before computing the variances, the lines are partitioned into groups based upon their 2-D image plane orientations. Equation 5.2 is then computed only for lines belonging to different orientation groups.

5.3.2 Line Orientation and Camera Rotation

In this section, a procedure is given for estimating the angle of rotation about the gravitational axis. This algorithm uses the set of horizontal lines extracted in the previous section. Each 3-D line in image frame coordinates is related to its corresponding world coordinate line by

$$d_s = R_f d_{fs} \quad (5.3)$$

where R_f is a rotation matrix. This matrix can be decomposed into two matrices

$$R_f = R_{\phi_f} R_{g_f} = [i_f j_f k_f] \quad (5.4)$$

where R_{g_f} is a rotation about the gravitational axis, and R_{ϕ_f} is a rotation which aligns the Y-axis of image frame f with the 3-D gravity vector g_f . If we assume that the world coordinate system Y-axis is aligned with gravity, then R_{ϕ_f} can be written as

$$R_{\phi_f} = \begin{bmatrix} \cos \phi_f & 0 & -\sin \phi_f \\ 0 & 1 & 0 \\ \sin \phi_f & 0 & \cos \phi_f \end{bmatrix} = \begin{bmatrix} C_f & 0 & -S_f \\ 0 & 1 & 0 \\ S_f & 0 & C_f \end{bmatrix} \quad (5.5)$$

where ϕ_f is the unknown parameter of rotation for frame f . R_{g_f} is a rotation which aligns j_f with g_f . If we also assume that R_{g_f} does not rotate i_f out of the plane defined by i_f and g_f , then this matrix can be solved for in terms of the gravity vector g_f .

$$R_{g_f} = \frac{1}{\sqrt{1-g_{f_x}^2}} \begin{bmatrix} 1-g_{f_x}^2 & -g_{f_x}g_{f_y} & -g_{f_x}g_{f_z} \\ -g_{f_x}\sqrt{1-g_{f_x}^2} & -g_{f_y}\sqrt{1-g_{f_x}^2} & -g_{f_z}\sqrt{1-g_{f_x}^2} \\ 0 & g_{f_z} & -g_{f_y} \end{bmatrix} \quad (5.6)$$

From the above definitions it can be seen that Equation 5.3 is a set of 2 linear equations and 4 unknowns. The line d_s provides 2 unknown parameters, and the dependent rotational parameters S_f and C_f from Equation 5.5 make up the addition 2 parameters. This set of equations could be solved from a single line viewed in 2 image frames. A more robust formulation is to minimize the sum of squares of Equation 5.3

$$\sum_{f=1}^n \sum_{s=1}^m [R_f d_{fs} - d_s]^T [R_f d_{fs} - d_s] \quad (5.7)$$

where n is the number of image frames and m is the number of horizontal lines. This equation is a quadratic form and can be minimized by solving for the eigenvectors of the associated symmetric matrix. This matrix is order $2(n+m)$. The eigenvector corresponding to the smallest eigenvalue gives the least squares solution.

5.3.3 Line Position and Camera Translation

The final step in line reconstruction is to solve for the line positions relative to the world coordinate system. Previously, the line positions were found by positioning the origin of the world coordinate system at the focal point of image frame 1, and then arbitrarily choosing a point on each line in this first image frame. The position estimates produced by this camera centered coordinate system were sensitive to noise because of the small distance between points, in comparison to the distance of the points from the origin.

A new formulation for this problem was produced, which uses an object centered coordinate system to solve for the line positions. Each point p_s is constrained to lie within the plane perpendicular to $R_f n_{fs}$. This plane can be expressed as

$$n_{fs}^T R_f^T [p_s - t_f] = 0 \quad (5.8)$$

where t_f is the translation of image frame f in world coordinates. To position the origin at the center of an object, two additional constraints will be imposed. The first is that the points must be perpendicular to their corresponding lines.

$$d_s^T p_s = 0 \quad (5.9)$$

The second constraint is that the sum of all the points equal zero.

$$\sum_{s=1}^m p_s = 0 \quad (5.10)$$

These three sets of equations are sufficient to solve for the line positions p_s , as well as the translation of each image frame t_f . Once again the sum of squares can be minimized for a robust solution.

$$\sum_{s=1}^m \left[\sum_{f=1}^n [p_s - t_f]^T R_f n_{fs} n_{fs}^T R_f^T [p_s - t_f] + p_s^T d_s d_s^T p_s + p_s^T p_s \right] \quad (5.11)$$

The matrix of the quadratic form is order $3(n+m)$. The eigenvector corresponding to the smallest eigenvalue gives the least squares solution for the parameters p_s and t_f . The object centered solution is slightly more computationally expensive than the camera centered solution which is order $3n+m$. However, the results are significantly more robust.

5.4 Error Analysis

The algorithm presented in this paper was implemented and tested on several sequences of noise corrupted synthetic data. The performance of the algorithm was tested with respect to errors caused by image resolution and errors in the gravitational axis. The algorithm was found to be robust with respect to errors in the 2-D line orientations. It was more sensitive to errors in the gravitational axis, but was found to be tolerant of these errors as well. The effects of image resolution on the computation of the rotational angles is examined in the following section. Section 5.4.2 shows the effects of a noisy gravitational axis on the determination of these angles. Finally, the derived motion parameters are used to reconstruct the environmental structure. These results are presented in Section 5.4.3.

5.4.1 Errors from Reduced Image Resolution

Images are discrete, therefore it is important to quantify the effects of image resolution on any computer vision algorithm. In the following experiments, noise was introduced by constructing a 20 pixel normal for each line. This normal was then rounded off to the nearest pixel, thus changing the 2-D line orientation.

Three frames from a fifty image sequence are shown in Figure 5.2.

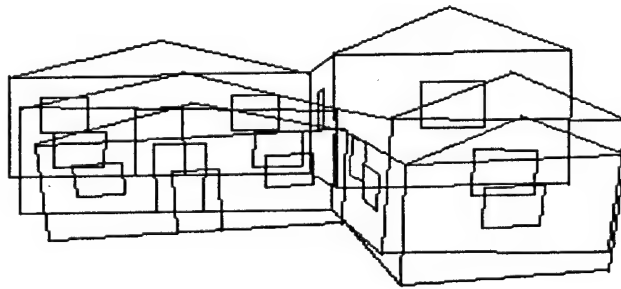
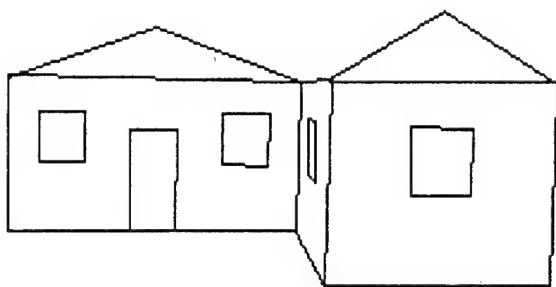


Figure 5.2. Frames 1, 25, and 50 from a 50 image sequence.

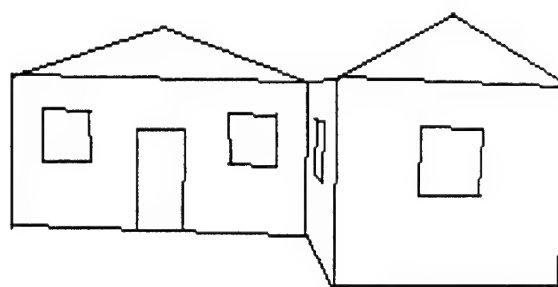
This data was produced by random rotations and translations. The image frames shown in Figure 5.2 are from a 512x512 image.

Images of size 512x512, 256x256, 128x128, and 64x64 are shown in Figure 5.3.

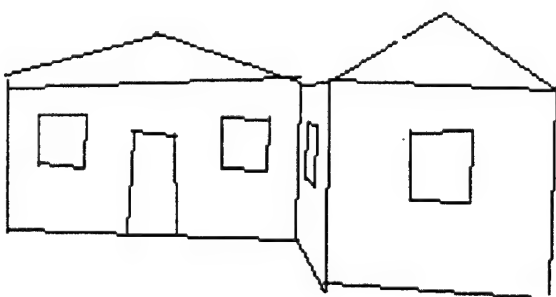
512x512 Image



256x256 Image



128x128 Image



64x64 Image

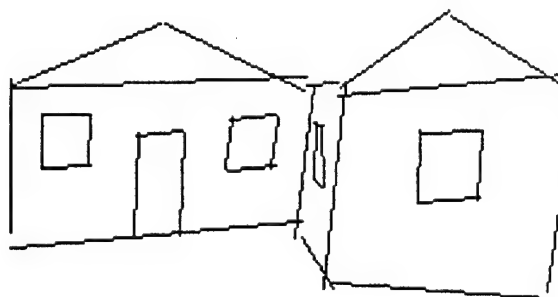


Figure 5.3. A single image frame produced using different image sizes.

The algorithm was applied to the motion sequence shown in Figure 5.2 with varying image sizes. The computed angles of rotation ϕ_f associated with these different image sizes are shown in Figure 5.4.

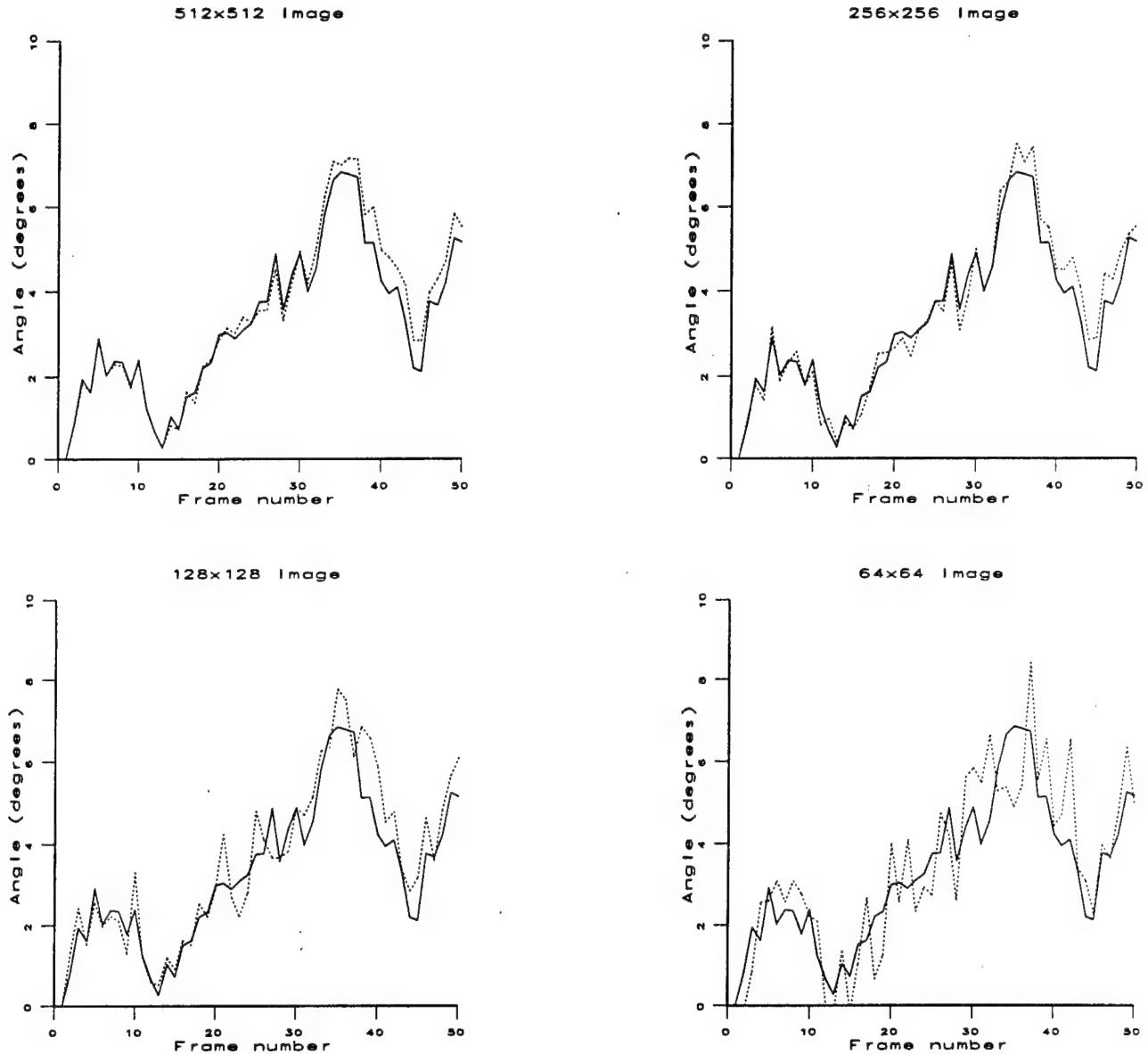


Figure 5.4. Angle of rotation computed from different image sizes. The correct angles are shown with solid lines, and the derived angles are shown with dotted lines.

The solid lines correspond to the correct values, and the dotted lines correspond to the derived values. The maximum error for the 512x512 image sequence was 0.874° and the average error was 0.274° . The 256x256 images had a maximum error of 0.783° and an average error of 0.309° . The 128x128 images had a maximum error of 1.724° and an average error of 0.516° . Finally, the 64x64 images had a maximum error of 2.431° and an average error of 0.842° . These

results demonstrate the robustness of this algorithm with respect to the errors caused by reduced image resolution.

5.4.2 Gravity Measurement Errors

In addition to image noise, noise was added to the 3-D gravity vector by rotating it about a randomly generated axis. The amount of rotation was a uniformly distributed random value. The algorithm was applied to identical motion sequences with an image size of 512x512 and 128x128. The error in the gravity vector was a uniformly distributed random variable with maximum values of 1 degree and 4 degrees. The computed angles of rotation ϕ_f associated with these gravitational axis errors are shown in Figure 5.5.

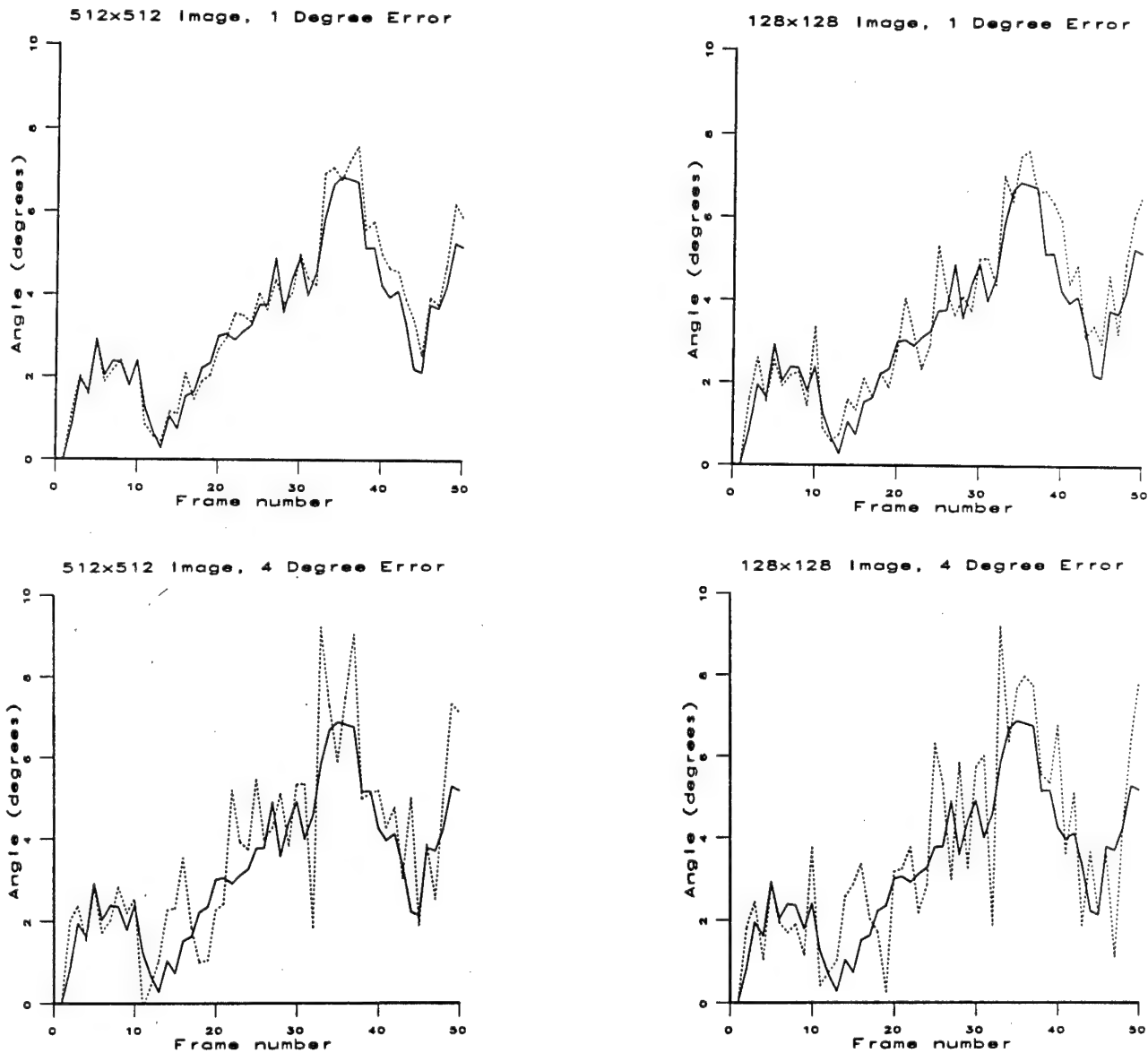


Figure 5.5. Angle of rotation computed from a noisy gravitational axis. The correct angles are shown with solid lines, and the derived angles are shown with dotted lines.

Again, the solid lines correspond to the correct values, and the dotted lines correspond to the derived values. The maximum error for the 512x512 image with 1 degree axis errors was 1.183°. The average error was 0.355°. The 128x128 image with 1 degree axis errors had a maximum error of 1.693° and an average error of 0.605°. The 512x512 image size coupled with 4 degrees axis error had a maximum error of 3.365° and an average error of 0.929°. The 128x128 image size with 4 degrees axis error had a maximum error of 3.340° and an average error of 1.066°. Once again the algorithm has proven to be robust in the presence of noise. The algorithm degrades gracefully until image and gravitational noise interfere with the determination of the horizontal lines. Once this happens, the results are no longer reliable. For the motion sequence presented here, the limits of the algorithm occur at an image resolution of 64x64 and a gravity vector error of 4 degrees. At this point the algorithm correctly chose 2 of the 15 horizontal lines to use for the motion computation.

5.4.3 Recovery of Structure

Once the parameters of rotation are known, the environmental structure can be recovered as discussed in Section 5.3.3. The recovery of structure will be demonstrated for the 512x512 and 128x128 image sequences using the results from Figures 5.4 and 5.5.

The reconstruction algorithm was first applied to the 512x512 image sequence shown in Figure 5.4. Figure 5.6 shows a top-down view of the reconstructed lines overlaid on the model.

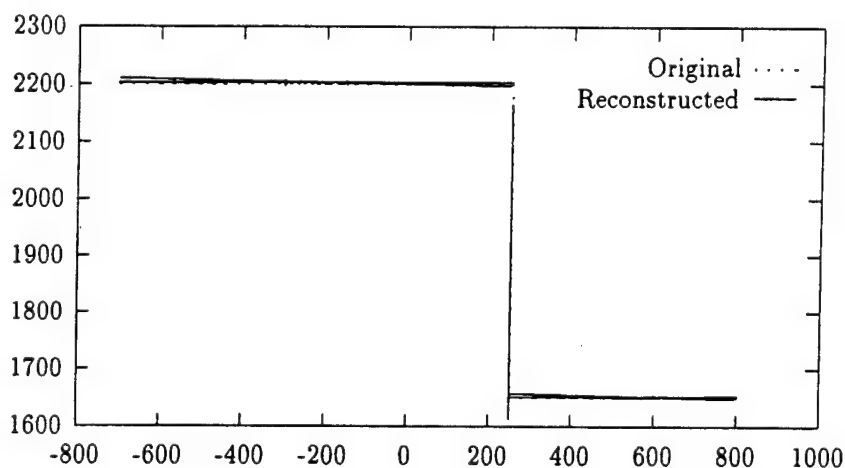


Figure 5.6. Top view extracted from 512x512 images and no gravity error.

The model is drawn with solid lines, and the reconstructed data is drawn with dotted lines.

The reconstruction from a 128x128 image sequence with no gravity errors is shown next in Figure 5.7.

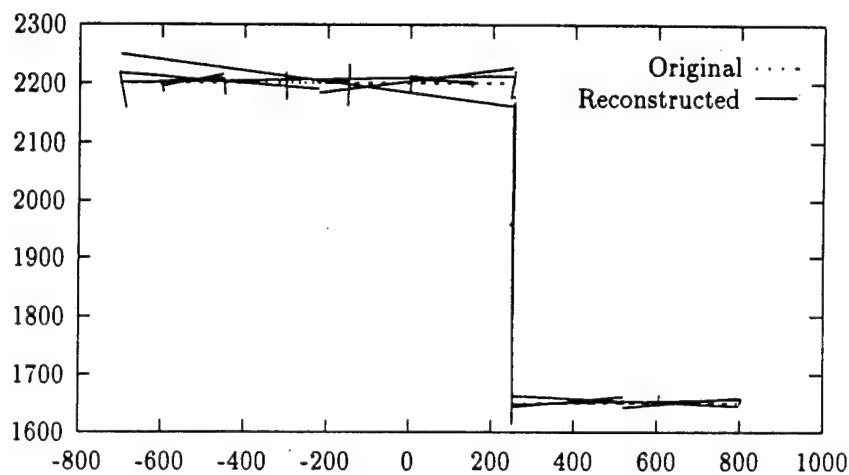


Figure 5.7. Top view extracted from 128x128 images and no gravity error.

From this reconstruction it is clear that the algorithm performs well with changes in image resolution.

The reconstruction algorithm was again applied to the 512x512 image sequence. This time the gravity vector was corrupted up to 4 degrees as shown in Figure 5.5. The reconstructed data is shown in Figure 5.8.

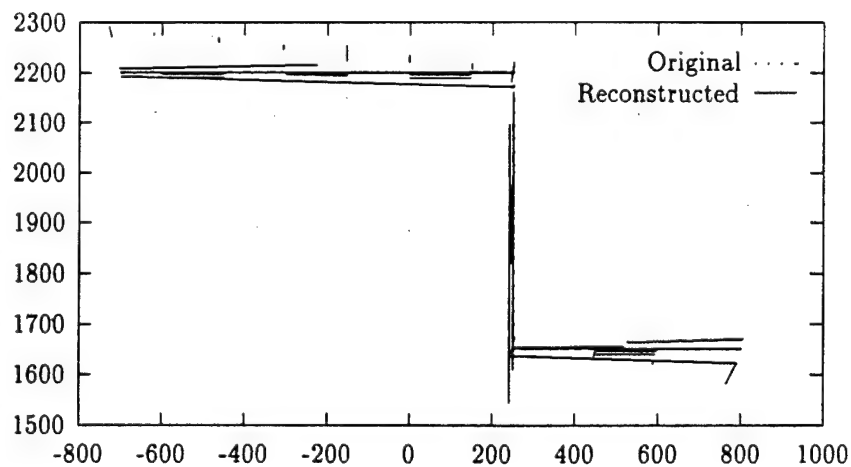


Figure 5.8. Top view extracted from 512x512 images and 4 degrees gravity errors.

The final reconstruction was performed using the 128x128 image sequence and up to 4 degrees of error in the gravity vector. The solution for the angle of rotation for this sequence is shown in Figure 5.5. Figure 5.9 shows the reconstructed lines overlaid on the original model. From these depth results it is clear that this algorithm is robust and can perform well under noisy conditions.

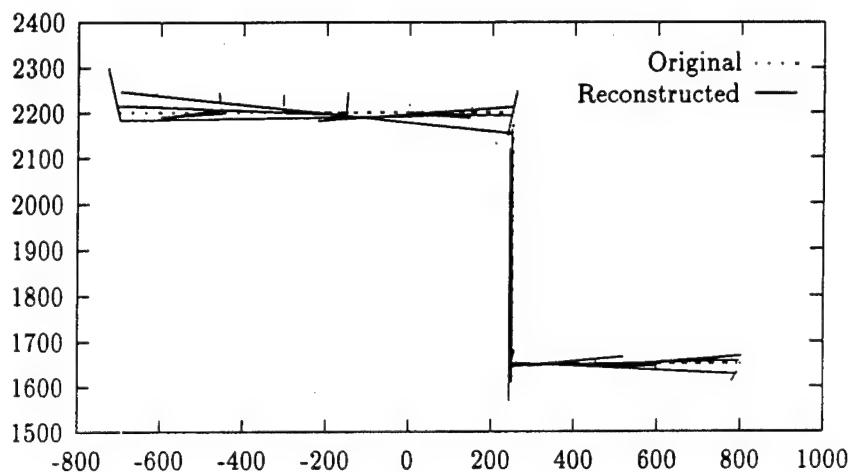


Figure 5.9. Top view extracted from 128x128 images and 4 degrees gravity error.

5.5 Results

The algorithm presented in this chapter was tested on an image sequence taken from a hand-held camera. Figures 5.10, 5.11, and 5.12 are the 1st, 25th, and 50th frames respectively of a 50 image sequence. Each image in the sequence is 640x480 pixels. The camera motion was arbitrary.



Figure 5.10. 1st image from a 50 frame sequence.

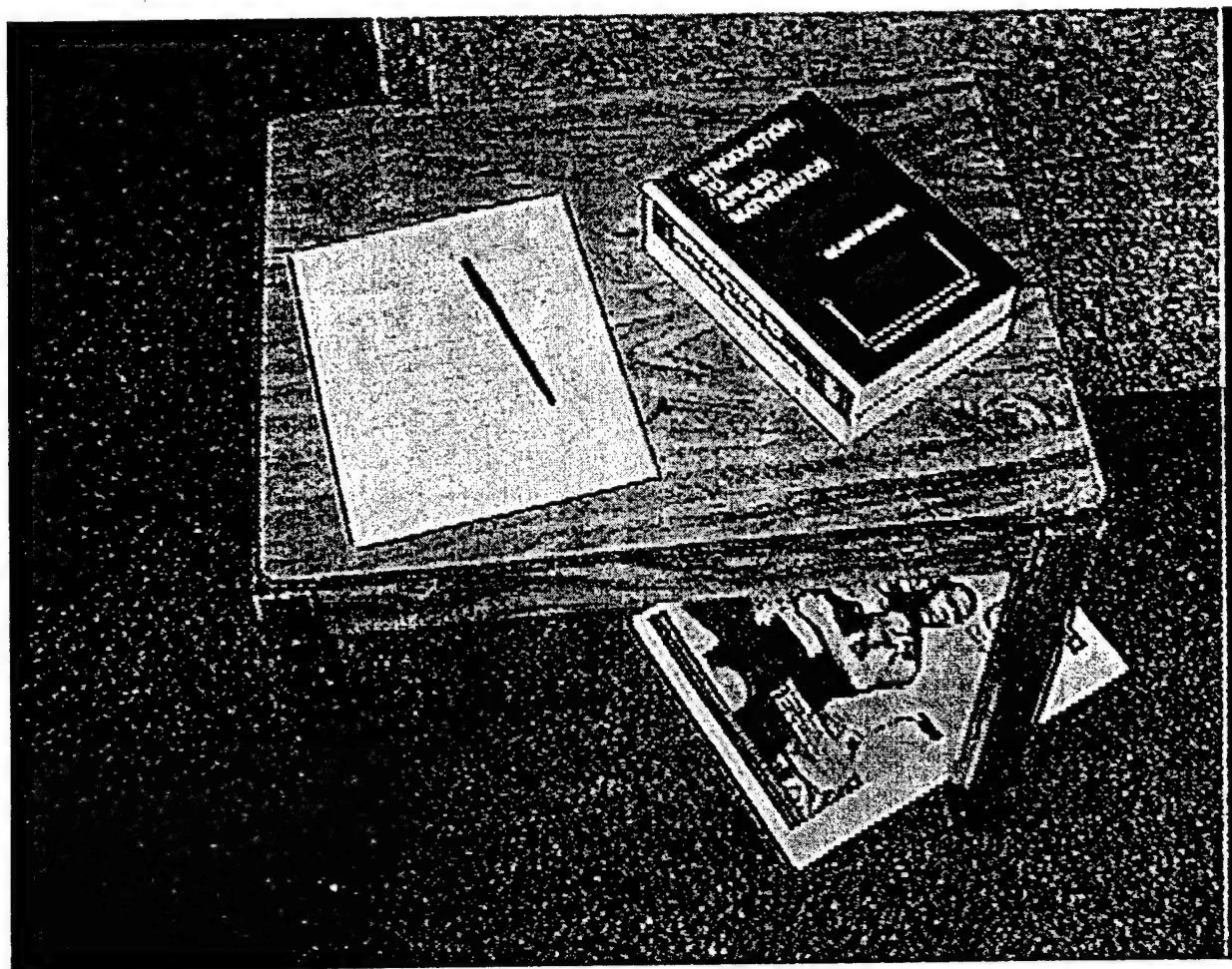


Figure 5.11. 25th image from a 50 frame sequence.

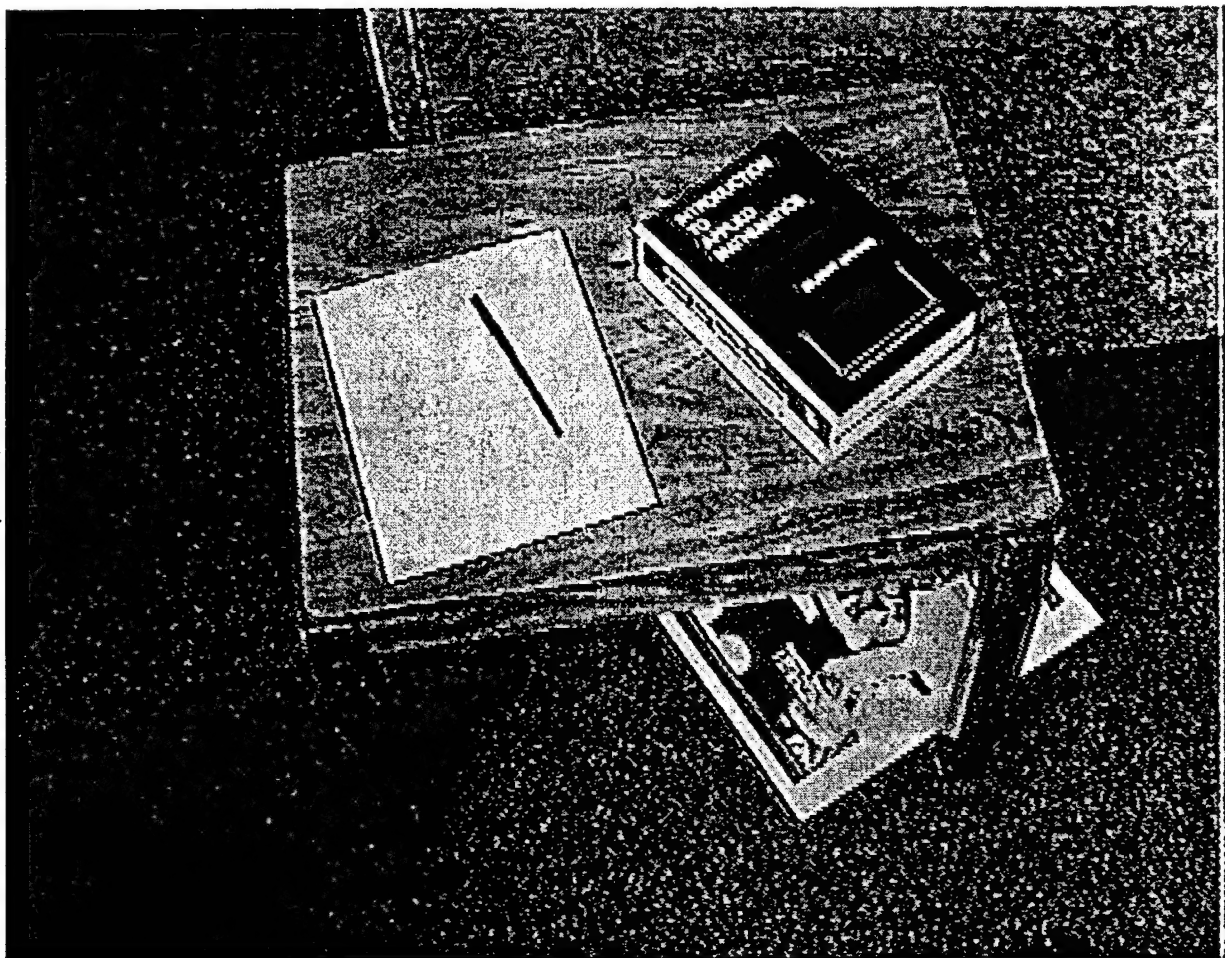


Figure 5.12. 50th image from a 50 frame sequence.

5.5.1 Line Extraction and Matching

Lines were extracted from this image sequence through the use of the Hough transform. Initially, the images were smoothed with a low-pass filter. The low-pass filter decreased the resolution of the images to 400x300 and then expanded them back to the original resolution. The images were then convolved with 3x3 sobel masks to produce gradient images. The gradients were thresholded based upon their magnitudes, and then non-maxima suppression was employed to thin gradient areas. The resulting gradient images were used as the inputs to the Hough transform. Figure 5.13 shows the first image frame after non-maxima suppression.

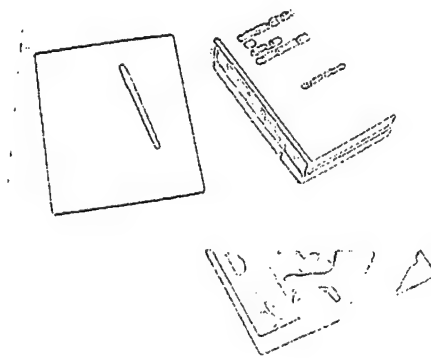


Figure 5.13. Image 1 after non-maxima suppression.

The non-maxima image was then transformed into a 2-D Hough space. One axis of the Hough space corresponds to the gradient direction. The other axis corresponds to the position of this gradient. The position is defined as the minimum distance to the line orthogonal to the gradient. Figure 5.14 is the distribution in this Hough space of the gradient image shown in Figure 5.13.

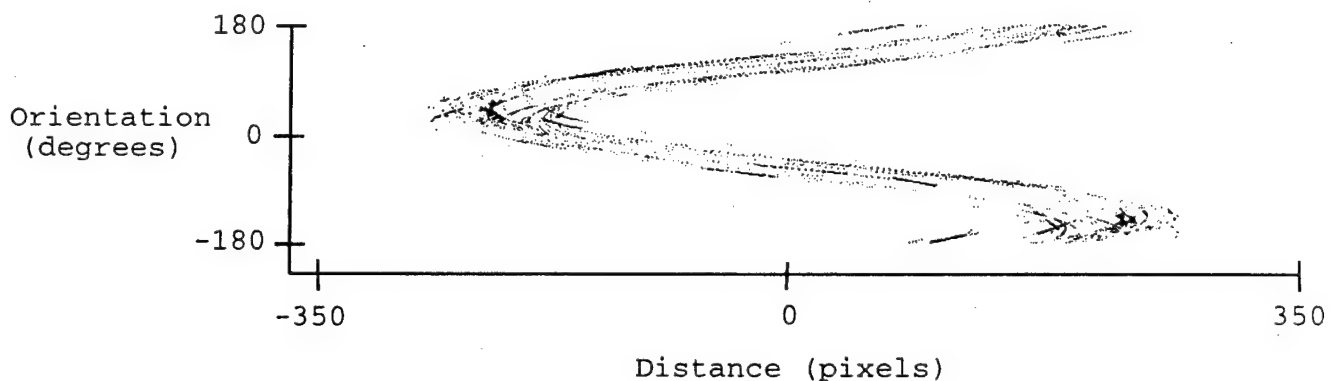


Figure 5.14. Hough space corresponding to image 1.

Potential lines were identified by locating local maximums in the Hough space. Endpoints were determined by walking along each line in the non-maxima gradient image. Lines were split or merged based upon the position of the endpoints and the information obtained while line walking. The lines extracted by this line algorithm from images 1, 25, and 50 are shown in Figures 5.15, 5.16 and 5.17, respectively.

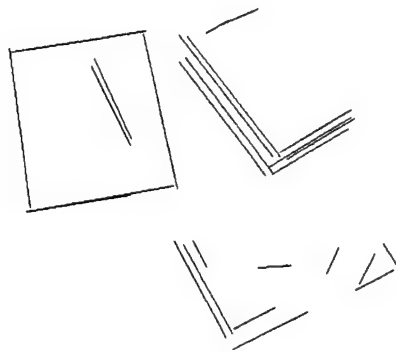


Figure 5.15. Lines extracted from image 1.

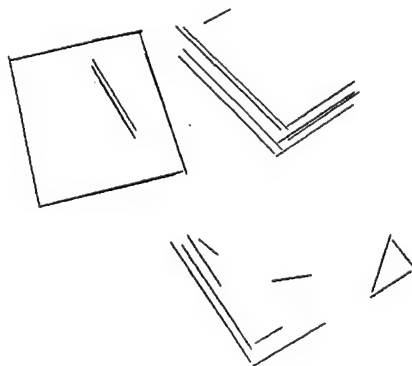


Figure 5.16. Lines extracted from image 25.

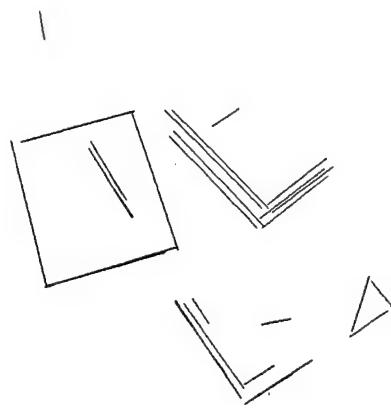


Figure 5.17. Lines extracted from image 50.

The lines extracted from each image were then matched between frames. Fourteen lines were selected manually in the first image frame, and then tracked automatically throughout the fifty frame sequence. The matching algorithm compares two skewed lines with all the lines in the next image. Each potential match for the two lines is evaluated by calculating a 2-D image plane rotation and translation, and then matching the remaining lines. An error value is used to determine the best match for each line, as well as the overall best set of matches. This error value is defined as the distance between the endpoints of a rotated and translated line, and its potential match in the next image.

5.5.2 Scene Reconstruction

The set of fourteen matched lines, as well as a set of gravity vectors, were used as the inputs to the algorithm described in this chapter. The gravity vectors were extracted manually by selecting two vertical lines in each image. Then the planes defined by these lines and the focal point of the camera were intersected. The focal length of the camera was unknown, but results varied little with changes in the focal length. The results presented here were derived using a focal length of 1050 pixels (field of view of 33.9 degrees).

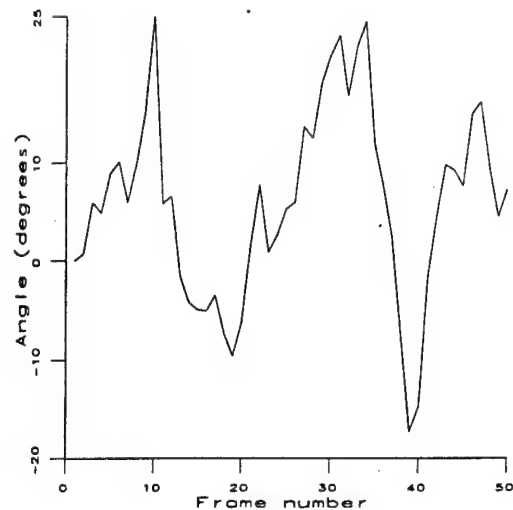


Figure 5.18. Derived angle of rotation.

Figure 5.18 shows the derived angle of rotation for the fifty frame sequence. These derived angles were then used to reconstruct the original scene. Figures 5.19 and 5.20 show the reconstructed 3-D lines with parts of image frame 1 texture mapped onto planes defined by the lines.

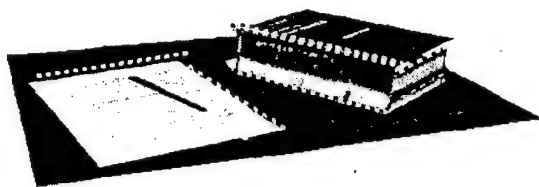


Figure 5.19. Perspective view #1 of the reconstructed scene.



Figure 5.20. Perspective view #2 of the reconstructed scene.

Figure 5.21 shows a closeup of the texture mapped books. The planes and texture coordinates were produced manually by selecting lines and picking regions from the image.

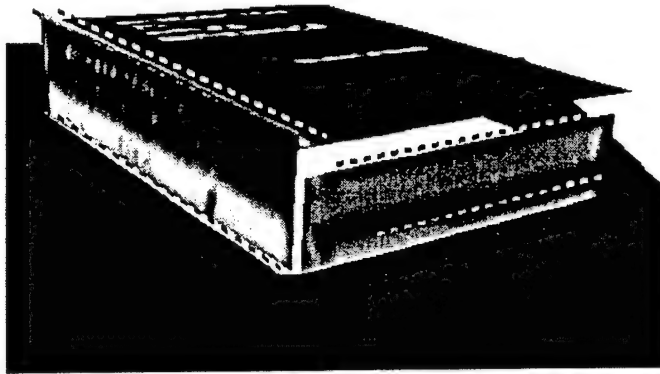


Figure 5.21. Perspective view of the books.

Since the motion undergone by the camera was arbitrary and unknown, the results must be examined with respect to the scene structure. The objects in the original scene were measured for this purpose. The reconstructed model also had to be scaled in order to make distance comparisons. The appropriate scale factor was determined by setting the width of the paper lying on the table to the correct value. Table 5.1 shows the angle and distance errors with respect to the estimated texture planes. The distances were measured from the midpoints of the reconstructed lines.

Table 5.1. Reconstructed line measurements.

Object	Number of Lines	Angle Error		Distance Error	
		Average	Maximum	Average	Maximum
Book Cover	3	0.000°	0.001°	0.813 cm	1.219 cm
Book Binding	2	0.041°	0.043°	0.000 cm	0.000 cm
Book Edge	2	0.200°	0.202°	0.000 cm	0.000 cm
Table Top	7	0.032°	0.063°	0.598 cm	1.560 cm
Floor	3	0.005°	0.009°	1.649 cm	2.473 cm

The estimated table top was then compared to the other surfaces in the scene in Table 5.2. The inter-plane distances were calculated by finding the distance along the normal from the center of the given plane.

Table 5.2. Reconstructed plane measurements.

Object	Angle with Table Top		Distance from Table Top	
	Value	Error	Value	Error
Book Cover	0.088°	0.088°	5.308 cm	0.565 cm
Book Binding	90.454°	0.454°		
Book Edge	88.030°	1.790°		
Floor	0.028°	0.028°	51.307 cm	1.300 cm

The estimated table top was then compared to the other surfaces in the scene in Table 5.2. The inter-plane distances were calculated by finding the distance along the normal from the center of the given plane.

Table 5.2. Reconstructed plane measurements.

Object	Angle with Table Top		Distance from Table Top	
	Value	Error	Value	Error
Book Cover	0.088°	0.088°	5.308 cm	0.565 cm
Book Binding	90.454°	0.454°		
Book Edge	88.030°	1.790°		
Floor	0.028°	0.028°	51.307 cm	1.300 cm

Chapter 6

Model-Based Vehicle Tracking

Efforts to develop intelligent and autonomous systems for operation in complex, natural domains have been largely unsuccessful to date, in spite of continued advances in the underlying technologies. There remain unresolved and fundamental difficulties in terms of the necessary computational power, the required complexity of perceptual systems which can operate in outdoor environments, and the corresponding complexity of planning and reasoning systems. A recent framework addresses many of these problems by stressing the importance of telerobotic and interactive systems [18,19]. This is a realistic approach to fielding advanced technology in the short term, and also provides a long term framework for developing autonomous systems. An interactive semi-autonomous system can significantly amplify the capabilities of a human, and also yields an evolutionary approach as autonomous system capabilities are developed and begin to replace human controlled functions.

The approach described here was to develop a model-based vision system that a human can interactively control. The inspiration for this system came from the interactive vehicle tracking system described in [11]. The human is responsible for building a 3-D model of the world by instantiating object models. These models are maintained by the vision system and are used to constrain future processing. Communication between the vision system and the human can then take place in the context of this shared model of the world which makes possible infrequent, semantically meaningful, low bandwidth communication.

The particular system presented is for tracking vehicles in outdoor scenes. A human can manipulate models of objects, such as gravity, roads, and vehicles, to aid in the interpretation of imagery from a camera. Once an interpretation is in place, the vision system can autonomously refine and extend the interpretations, detect and track vehicles, and report back to a human about unusual occurrences or behavior that cannot be accounted for. For example, a human will indicate that a particular area is a road, and another area is a vehicle. The system can then track the vehicle and determine if it goes off the road, or if it is behaving inconsistently with respect to the model of a vehicle.

This chapter begins with a discussion of the basic architecture of the interactive model-based vision system. This system is then discussed within the context of vehicle tracking.

6.1 System Architecture

The model-based system architecture is shown in Figure 6.1. The system is made up of two databases that a human can access and manipulate through a user interface. The human accesses models of the various types of objects stored in the Object Model Database to build an interpretation of the current scene which is stored in the World Model Database. As the user manipulates object models within the world model, the models are projected back against images for interactive control and to initiate processing.

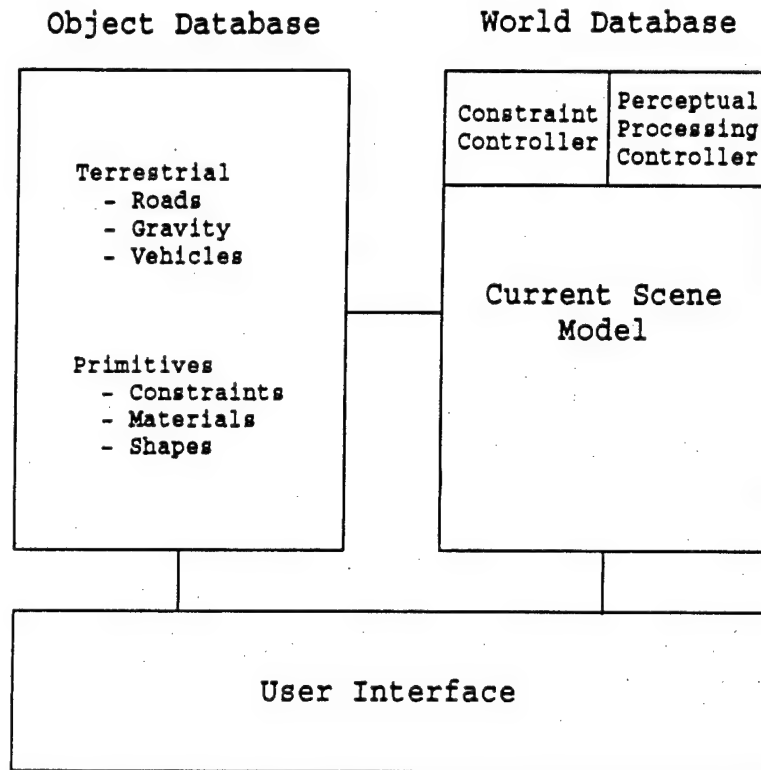


Figure 6.1. Model-based system architecture.

The Object Model Database contains generic models of objects, relationships, and events for terrestrial scenes. This database consists of objects such as roads, vehicles, and gravity. These objects can be divided into two categories: *terrestrials* which correspond to conventional objects found in the world, such as roads and cars, and *primitives* which correspond to basic entities and relationships which make up terrestrial objects. Primitive objects describe characteristics such as shape constraints, material composition, and relationships between parts. Thus, in addition to describing its shape, the model of a car needs to include the fact that a car is acted on by gravity, and will have a preferred type of orientation and attachment with respect to the ground surface. Object models are described by sets of constraints [2, 8, 14, 16] which must be satisfied. A simple constraint is that the value of some parameter associated with an object model is bounded. More complicated constraints deal with relations between objects. The human will in general specify a limited amount of information for an object, and the system will use the constraints and associated processing actions to then refine the instantiation of an object.

The World Model Database describes the 3-D world of objects and situations surrounding the vision system. It is initially formed by the human accessing models in the object database and instantiating them. There are two types of controllers associated with the World Model Database. The *constraint controller* checks for consistency in the world model. The constraint controller uses the constraints which define an object or relationship to refine an instantiation, or to find a violation or inconsistency and ask the human for help. The *perceptual processing controller* deals with the extraction of information from images. The constraints in an object model specify the types of processing that are necessary to obtain this information. For example, when a user indicates the location of a road, this constrains the type of tracking and feature extraction processes that are used.

The user interface consists of images, with object models displayed as graphical overlays. When the user instantiates an object model, the model is projected back against the image allowing the user to interactively manipulate the object within the world model.

6.2 Object Models

The interactive model-based system described in the previous section was applied to the domain of vehicle tracking. This domain contains complex lighting conditions and independently moving objects, yet there also are constraints which can be exploited through temporal and geometric models. Three models were developed to enable the tracking of vehicles in outdoor environments. The three types of objects modeled were roads, gravity, and vehicles.

6.2.1 Road Model

Knowledge of the position of a road is obviously beneficial to a vehicle tracking system. Information about road locations can restrict the amount of processing necessary to detect and track vehicles. The road model consists of a sequence of 2-D or 3-D line segments. A road is instantiated by drawing this sequence of lines on top of an image. This can be done by picking points with a mouse, or by tracking a vehicle moving along the road. Figure 6.2 shows a 2-D road model that was instantiated by a user with a mouse.

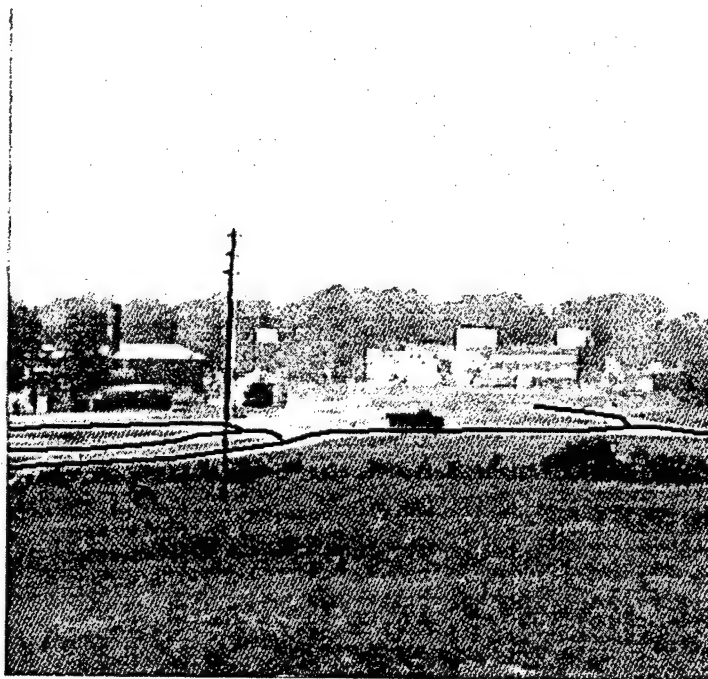


Figure 6.2. 2-D road model.

6.2.2 Gravity Model

Knowledge of the direction of gravity is useful in constraining the orientations of other objects in the world model. The position and orientation of falling objects are obviously affected by gravity.

Man-made objects, such as buildings, lie parallel to the direction of gravity, and vehicles are constrained to travel at angles close to the plane perpendicular to the gravity direction. This last constraint was exploited in this vehicle tracking system.

The gravity direction is represented by a 3-D vector. This direction is presented graphically in two ways. The 3-D vector is projected onto an image at different positions, creating the 2-D gravity field shown in Figure 6.3.

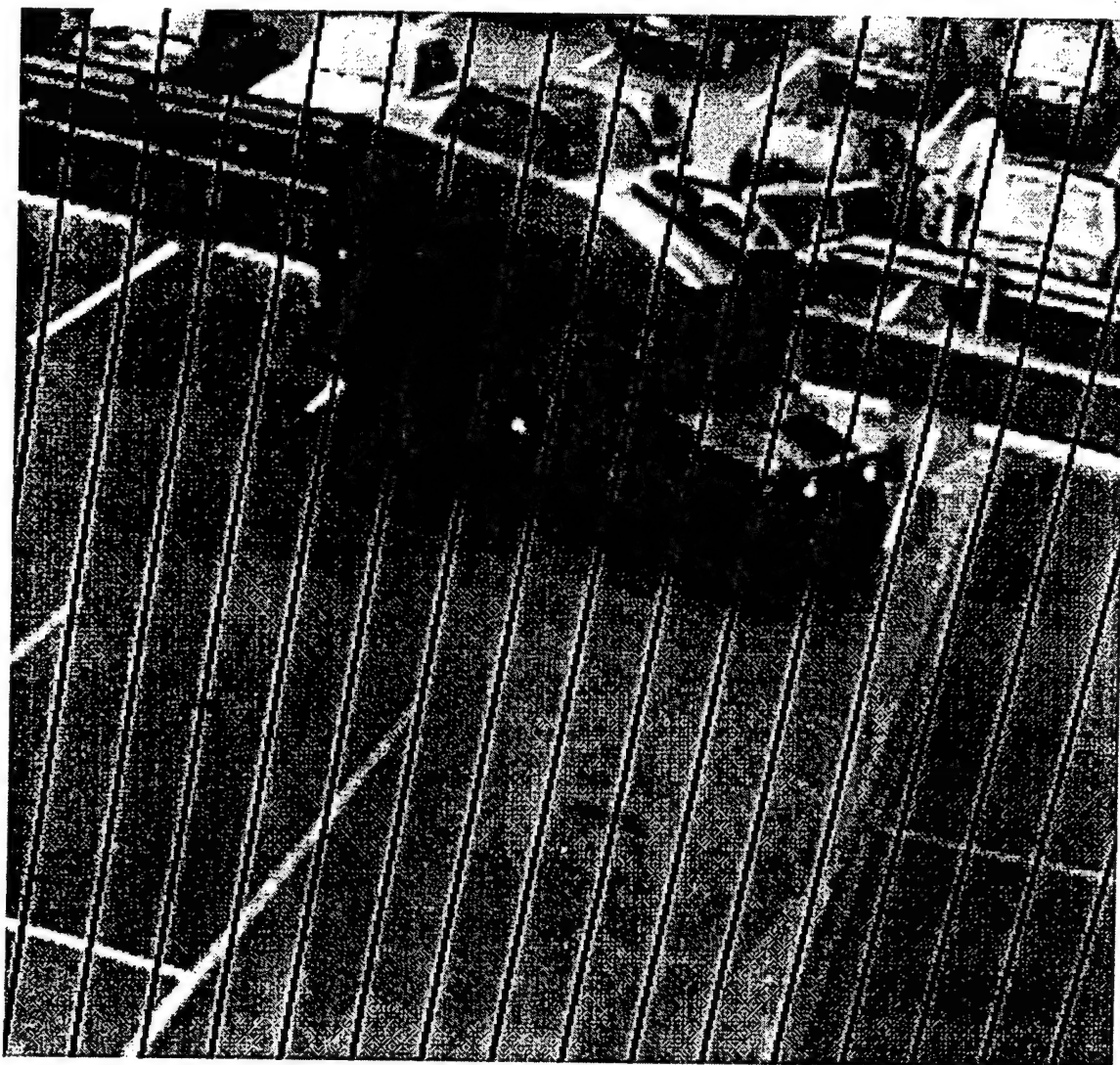


Figure 6.3. Gravity model projected onto an image.

The gravity vector is also displayed in a separate window as shown in Figure 6.4.

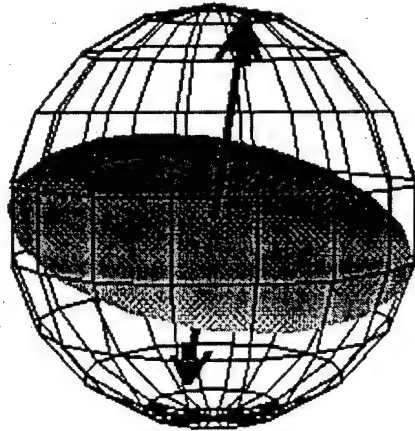


Figure 6.4. User interface for the gravity model.

The vector is given a planar base and drawn within a sphere to aid the user in determining the current 3-D position of the vector. A vector in the direction opposite gravity is also drawn in a lighter color to aid the user in determining the current gravity direction when the planar base interferes with the view of the gravity vector. This second representation also acts as the user interface for the gravity model. A user positions the gravity vector within the sphere using a mouse. As the user moves the vector within the sphere, the gravity field overlaying the image reflects the change in direction.

6.2.3 Vehicle Model

The most important model for a vehicle tracking system is certainly the vehicle model. The vehicle model geometry consists of six parameters: length, width, height, hood length, hood height, and wheel height. A perspective view of the vehicle model is shown in Figure 6.5.

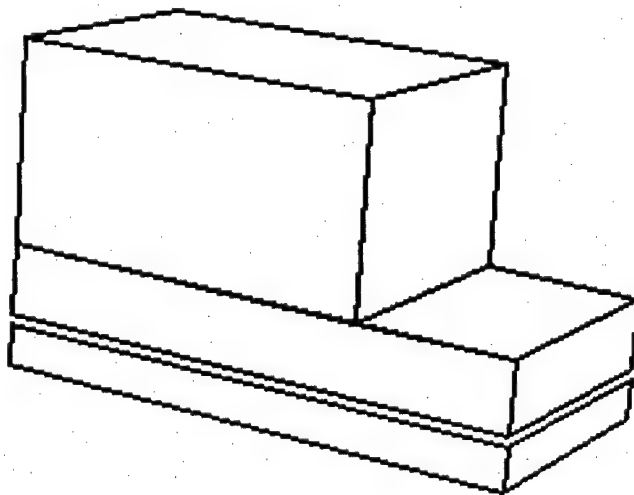


Figure 6.5. Perspective view of the vehicle model.

A user instantiates a vehicle within the world by placing this graphical model in the image. The six degrees of freedom of the graphical model can be manipulated using a mouse. Grabbing the vehicle with the left mouse button pressed, the user can move the vehicle parallel to the image plane. Pressing both left and middle buttons allows the user to move the vehicle perpendicular to the image plane, and pressing only the middle button allows the user to rotate the vehicle model.

6.3 Difference Tracker

The type of image processing or vehicle tracking that is employed is dependent upon the models that are instantiated. The current processing routines consist of two types of trackers and restricted segmentation and interest operators which are applied when models are instantiated -- a difference tracker presented in Section 6.3, and a Local Translation Tracker discussed in Section 6.4. One type of tracker is dependent upon a 2-D or 3-D road model, and can yield information allowing the automatic instantiation of a vehicle model. This tracker does not require the presence of these models, but with a road model this tracker is able to exploit the information provided by the model to construct a more efficient and intelligent solution to the tracking problem. This tracker is called a difference tracker, since it relies on information obtained from image differencing or subtraction.

The difference tracker operates with respect to an instantiated 2-D or 3-D road model. It determines regions above the indicated road areas which are changing over time and also are moving in a consistent direction. The tracker locates the front and rear of a vehicle and can use this information to instantiate a vehicle model. If a 3D road model has been instantiated, the location of the vehicle can be further constrained. The information recovered from this tracker can also be used to restrict the extraction of features for the local translation tracker discussed in Section 6.4.

The first step in tracking an object is to reduce the image noise by convolving consecutive images in a motion sequence with a low-pass filter. The images shown in this paper were smoothed using a Gaussian filter. If no models are present, the entire image must be convolved with this filter. However, given a 2-D road model, the filter is only convolved with pixels that are close to the road. The road model shown in Figure 6.2 is used to constrain the smoothing process, resulting in the image shown in Figure 6.6.



Figure 6.6. Image areas lying near the road model are smoothed.

Once the images have been smoothed, the algorithm begins to search for areas of motion that lie near the road. This is accomplished through image subtraction. Pixels from temporally consecutive images that are situated near the road model are subtracted. If the result of this subtraction is greater than a threshold, the environmental object corresponding to this pixel position is assumed to have undergone motion. This pixel is marked as a motion pixel, and a region growing process begins.

A 3-D road model with accompanying depth information would constrain the size of the projection of an object traveling along the road. However, without any depth, information the size of an object within the image is unknown. Since the size of the object within the image is unknown, the search for all areas of motion associated with an object is accomplished through region growing. Once a pixel near the road has been identified as a motion pixel, its neighbors are also examined

using the subtraction technique discussed above. If any of the neighboring pixels contain motion, their neighbors also are examined. This recursive procedure continues until no more motion pixels can be found. The areas of motion found using the first two images in a motion sequence are shown in Figure 6.7.

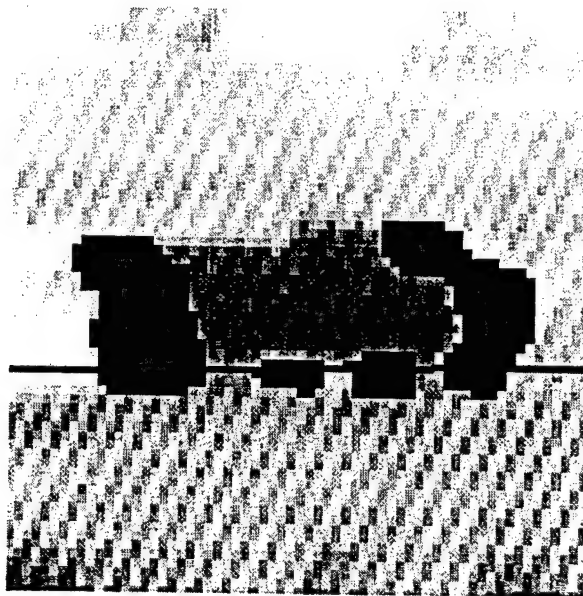


Figure 6.7. Areas of motion extracted by region growing.

In this figure the areas of motion are shown overlaid on a sub-image extracted from the first image in the motion sequence. The entire image is shown in Figure 6.2. Once the areas containing motion have been identified, the centroid of these areas is located. The motion pixels are fitted with a bounding box, and the center of this box is taken as the centroid as shown in Figure 6.8.

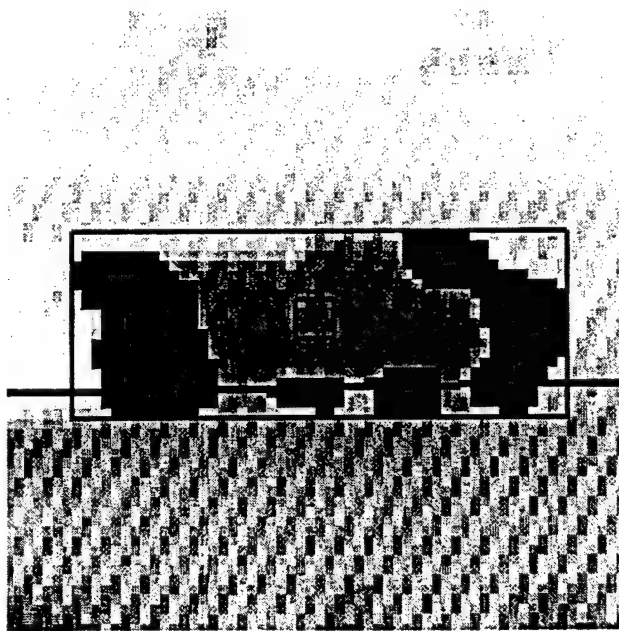


Figure 6.8. Center of the areas of motion.

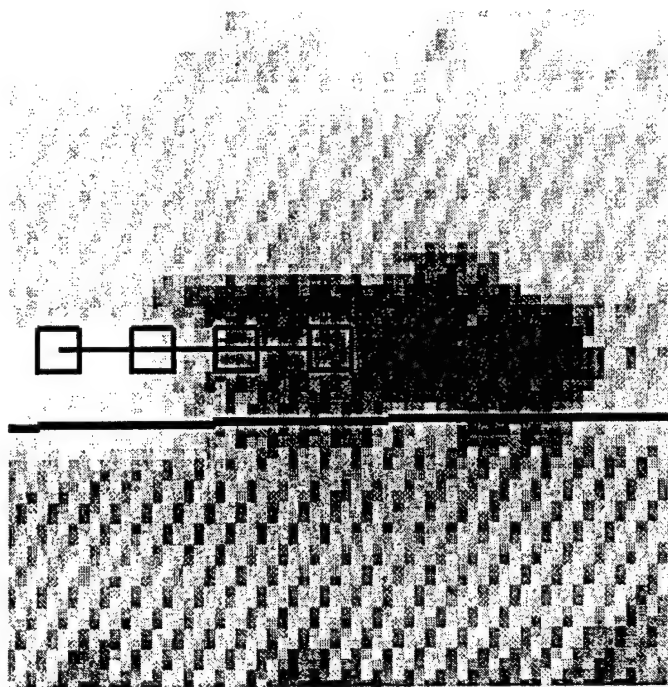


Figure 6.9. Two dimensional motion trajectory.

Over time a 2-D trajectory can be constructed. Figure 6.9 shows the results of processing a five image sequence. The trajectory is overlaid on the last image in the motion sequence. In the presence of a 3-D road model, the 3-D motion of the object can be inferred. The introduction of a simple, easily manipulated road model results in an efficient and simple tracker. This tracker can be used to scan road areas for any changes, notifying a user or instantiating a vehicle model when they occur. The instantiation of a vehicle model spawns the local translation tracker discussed in the next section.

6.4 Local Translation Tracker

Vehicles have a limited turning radius, which results in an axis of rotation that is often far away from the vehicle. A vehicle tracker was constructed based upon this constraint and using the concepts presented in [10]. The local translation-based vehicle tracking system architecture is shown in Figure 6.10.

A vehicle is subdivided into local regions, and each region is treated as if it had undergone purely translational motion. The extraction and grouping of features can be done automatically, but is more efficient and reliable when directed by a vehicle model. Instantiation of the vehicle model, shown in Figure 6.5, spawns the local translation tracker. This tracker consists of feature extraction and grouping, and 3-D trajectory computation via local translational decomposition and edge matching. The information derived from this tracker can be used to determine a 3-D road model, as well as refine the attributes of the instantiated vehicle model through temporal modeling with the Kalman Filter.

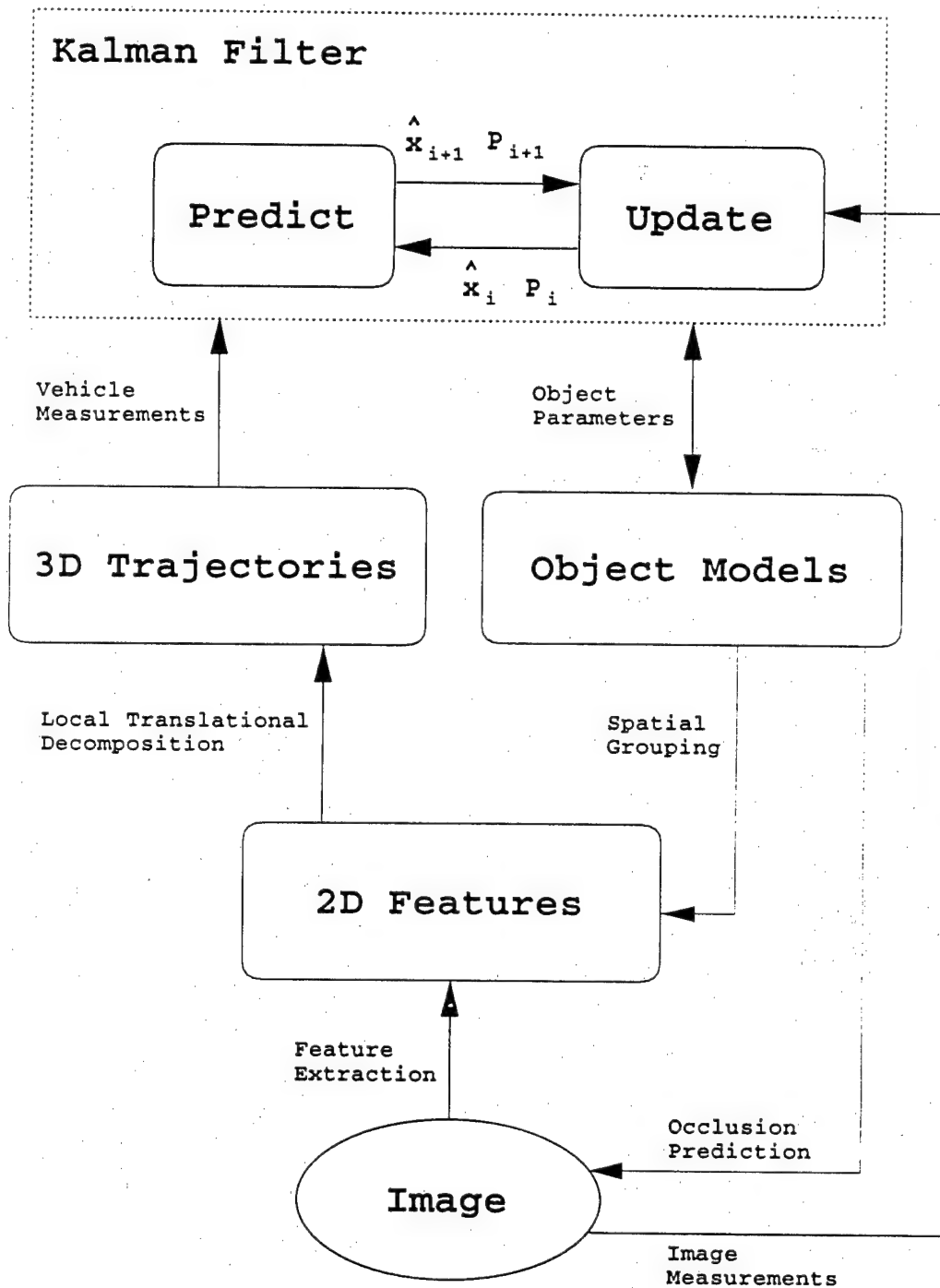


Figure 6.10. Vehicle tracking system architecture.

6.4.1 Feature Extraction

The local translation tracker requires features which can be matched in successive images. The type of features used are conventional masks of image pixels, extracted from distinct areas of the image. In the examples shown in this chapter, the masks are 9x9 pixel arrays. Normalized correlation is used to determine similarity of extracted features. Normalized correlation is used both in measuring feature distinctiveness, and for evaluating the matches of extracted features along the radial flow determined by a possible axis of translation. Since the radial flow lines do not necessarily pass through the center of the image pixel arrays, bilinear interpolation is used for matching features. This allows the extraction of masks and correlation to be performed at a continuous range of locations, rather than just at the discrete pixel positions, resulting in more accurate correlation values.

The distinctiveness of a feature is 1 minus the best correlation value obtained when the feature is correlated with its immediately neighboring areas. Good features are selected by finding the local maxima in the values of the distinctiveness measure over an image. Neighborhoods over which the features are selected are constrained to areas that contain large intensity discontinuities, determined by extracting zero-crossings. This generally results in the extraction of areas of high curvature along the zero-crossing contours. The areas of feature extraction can be further constrained by the output of the difference tracker or by an instantiated vehicle model. The distinctiveness measure is then applied only to these restricted areas in an image.

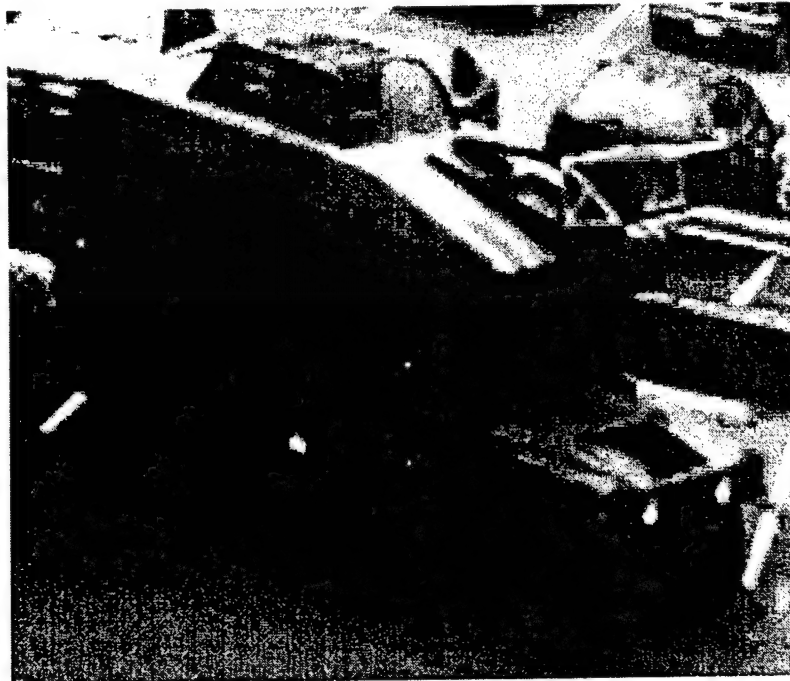


Figure 6.11. Image of a truck.

Figure 6.11 shows a portion of an image that contains a truck.

The zero-crossing contours are shown with the extracted features overlaid in Figure 6.12.

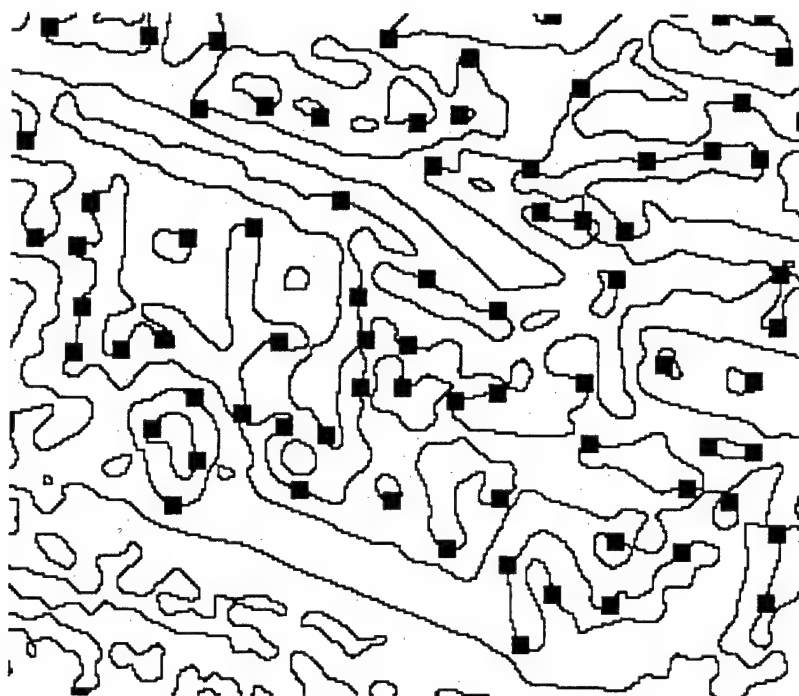


Figure 6.12. Zero crossings and features extracted from the truck image.

As a vehicle is tracked over a sequence of images, this processing is continually reapplied. New extracted features are the result of occlusions or changes in observable detail as a vehicle moves in depth.

6.4.2 Feature Extraction from a Model

The information provided by models can be used to direct feature extraction. The local translation tracker estimates the local translation of different portions of the vehicle, thus each portion requires a certain density of features for this estimation. Sequential images are registered by calculating a 2-D displacement parallel to the image plane. This requires only a small set of features over the entire image. The vehicle model is used to determine the appropriate density of features at each image position.

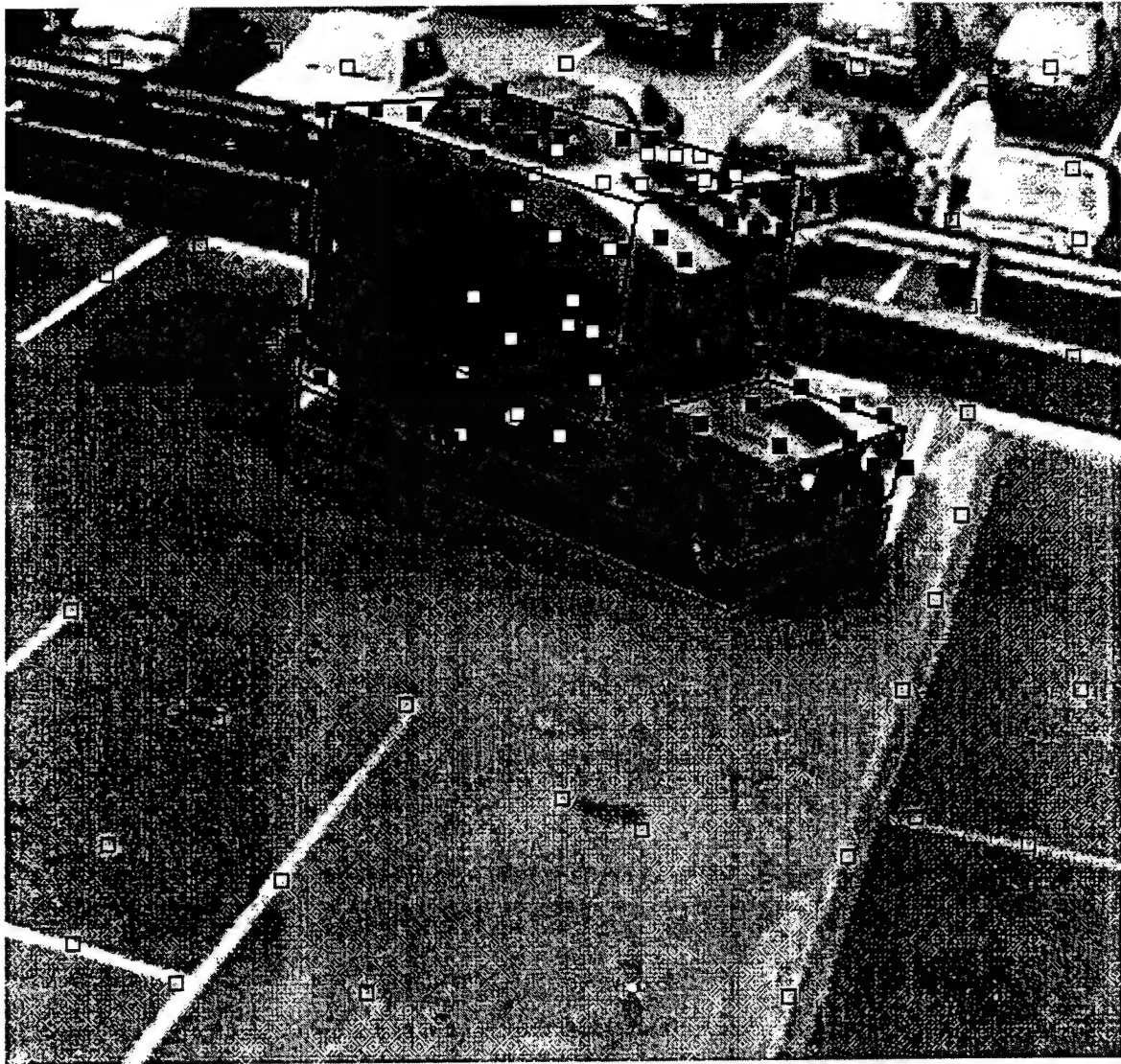


Figure 6.13. Features grouped using a vehicle model.

Figure 6.13 shows a set of features extracted from an image with a vehicle model. Notice that the density of features is much higher within the vehicle model where the features are used to calculate several 3-D displacements. Outside the vehicle the features are used to calculate a single 2D displacement, and so fewer features are necessary.

The vehicle model also is used to group the features lying on the vehicle. The local translation tracker computes a direction of translation at different points along the surface of the vehicle. Vehicle motion is locally planar, so the majority of rotation will be about the normal to this plane. Therefore, features that lie close together with respect to the length of the vehicle have similar trajectories. The local translation tracker groups features using this distance criterion. The results presented in this chapter were all produced by grouping the vehicle features into three groups: the front, center, and rear features. These groups are shown in Figure 6.13.

The vehicle model also is used to determine areas in which feature extraction should be performed because of occlusion. Areas in the background are occluded by the vehicle, and as the vehicle moves, these areas become visible and feature extraction is performed. Self-occlusion by the

vehicle is also detected using the vehicle model. In this case, vehicle rotation exposes previously occluded areas on the vehicle.

The feature positions used to derive the local translation vectors are not direct measurements of the position of the vehicle. It is necessary to measure quantities in the image that are directly related to the position of the vehicle for the purpose of initializing the vehicle position, and measuring the quality of subsequent estimated positions. Searching for vehicle features, such as headlights or wheels, is one way to measure the position of the vehicle. Currently, the edges of the vehicle model provide the necessary measurements.

A sub-image is constructed about each edge that is to contribute to the measurement. The size of this sub-image is determined using the covariance matrices associated with the Kalman Filter (see Section 6.4.4). The sub-image is then convolved with a one dimensional edge mask oriented in the direction of the edge. Each row of the resulting sub-image is then summed, and the maximum values are considered possible positions for the edge. The position of the vehicle can be determined given three edge positions. Currently, three edges are used to solve for the vehicle position, and then the error sum at a fourth edge is used to verify the solution. The position is calculated using the three maximum positions for each of the three edges, so twenty-seven possible positions are determined. The fourth edge is added to the error sum for each possible position, and the maximum value is chosen as the position of the vehicle.

An image with a user-instantiated vehicle model is shown in Figure 6.14

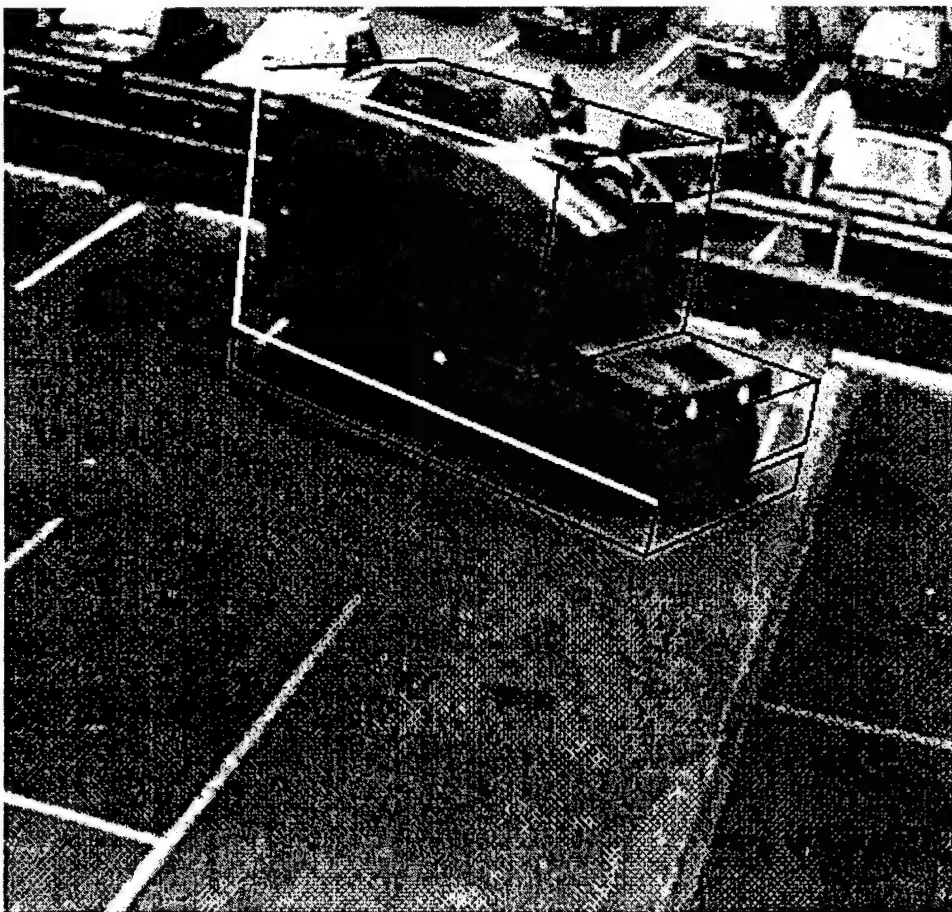


Figure 6.14. User-instantiated vehicle model.

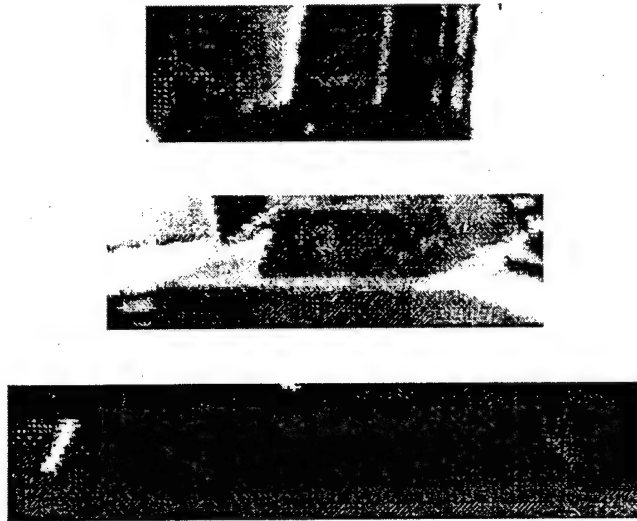


Figure 6.15. Sub-images extracted using the vehicle model.

The three edges used to find the vehicle position are white; the remaining edges are black. Figure 6.15 shows the three sub-images extracted using the vehicle model. The corresponding convolved images are shown in Figure 6.16.

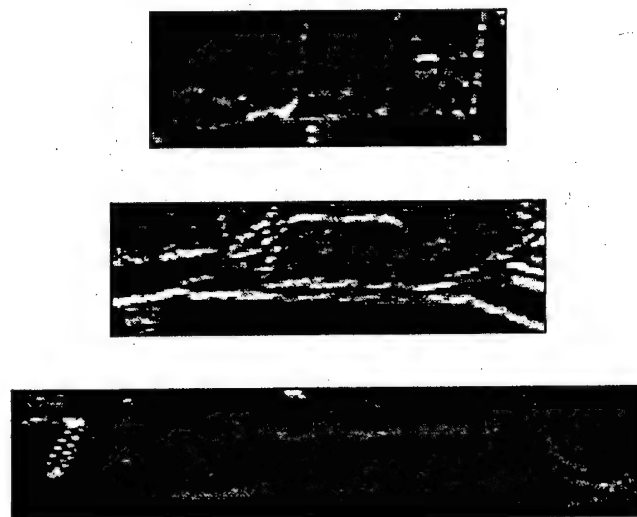


Figure 6.16. Convolved sub-images.

The top, middle, and bottom sub-images correspond to the rear, top, and bottom vehicle model edges. A new vehicle position was generated using the convolved images and is shown in Figure 6.17.

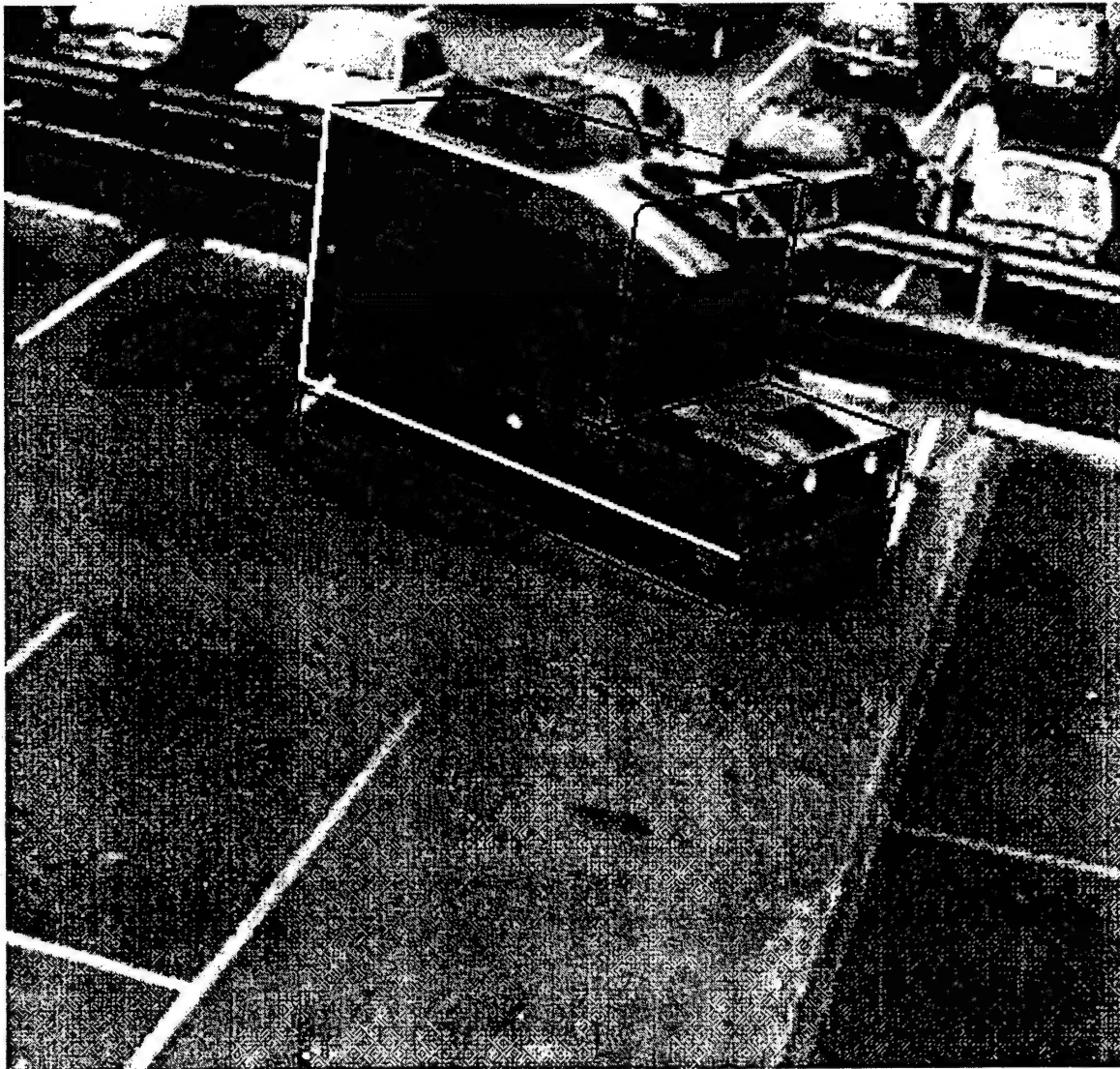


Figure 6.17. Adjusted vehicle model position.

6.4.3 Locally Planar Motion

Often the motion of a vehicle is restricted to a plane determined by the local road or surface orientation. In this case, it is possible to associate 3-D information with extracted image features. The directions of motion are constrained to be parallel to the given plane, so the possible directions of motion for the local translation tracker are restricted to a circle on the unit sphere whose orientation is parallel to the plane of motion. Locally planar motion also results in a smaller temporal filter.

The gravity model defines a plane which can be used to restrict the motion of the vehicle model. Instantiation of the gravity model places constraints on the orientation of the vehicle model. If a vehicle model is instantiated, gravity will initially be aligned with the vehicle axis. As the gravity direction is changed, the vehicle orientation also changes. When a vehicle model is manipulated in the presence of a gravity model, the user will be restricted to rotations about the gravitational axis.

6.4.4 Temporal Filtering

Once the user has instantiated a vehicle model, the geometric and kinematic parameters that make up the model can be derived. The geometric parameters are fixed in the case of a known vehicle model, and in the case of an unknown model are parameters that must be estimated. The model shown in Figure 6.5 contains five parameters: length, width, height, hood length, and hood height. This geometric model is fixed at the time of instantiation and is not estimated by the tracker.

The kinematic parameters are described by the five dimensional vector $x = [p_x, p_y, v, \phi, w]^T$. This kinematic model assumes that the motion is in a plane perpendicular to the direction of gravity. However, errors in the direction of gravity, as well as other modeling errors, results in motion that is not purely planar with respect to the instantiated gravity direction. In order to allow the vehicle to deviate from the plane, an additional parameter was added to the kinematic model. This parameter is the distance of the vehicle above the plane of motion. The height parameter h is independent of the other kinematic parameters x , so it is maintained separately from the Kalman Filter model. An assumption that h remains constant over time was used along with an associated standard deviation σ_h in order to construct a recursive mean square error estimator for h .

The kinematic parameters are initially assigned values which are obtained after the user positions the vehicle model, and the edge-based adjustment has refined this initial position. The position $[p_x, p_y, h]$ and orientation ϕ are inferred directly from the position and orientation of the vehicle model. The velocity parameters v and w are derived by finding the position $[p_x, p_y]$ and orientation ϕ of the vehicle in an additional image frame.

The measurements of vehicle motion are computed using both the edge-based adjustment discussed in Section 6.4.2 and local translation as described in [10]. Measurements of $[p_x, p_y, h]$ are obtained using the edge-based adjustment, and measurements of $[v, \phi, w]$ are obtained using the local translation method.

The location of the vehicle at time $i+1$ is predicted using the kinematic estimate provided by the Kalman Filter at time i . This predicted position is used to extract edge information and to perform the edge-based adjustment in image frame $i+1$. The adjusted vehicle position is then passed to the filter as the current position measurement. Local translational decomposition is used to derive the remaining kinematic parameters. Features extracted from a vehicle are divided into three groups based upon spatial position as shown in Figure 6.13. Each group is treated as a translating rigid body, and the local translation vector is calculated for this section of the vehicle. A weighted average of the three local translation vectors is taken as an estimate of the linear velocity v , the vehicle orientation ϕ , and the angular velocity w . The rear axis of the vehicle is located between the middle and rear feature groups. Since the vehicle rotates about the rear axis, the vehicle motion at this point corresponds with the vehicle orientation. Therefore, the linear velocity is obtained by averaging the magnitudes of the middle and rear local translation vectors

$$v = \frac{\|t_{middle}\| + \|t_{rear}\|}{2}$$

and the vehicle orientation and angular velocity is obtained by averaging the three local translation vector orientations

$$\phi = \frac{\theta_{front} + 2 \cdot \theta_{middle} + 5 \cdot \theta_{rear}}{8}$$

$$w = \frac{v}{d} \cdot (\theta_{front} - \theta_{rear})$$

where d is the distance in 3-D space between the front and rear feature groups. Using the information obtained from both the edge-based adjustment and the local translational decomposition, a 5-D measurement vector $z = [p_x, p_y, v, \phi, w]^T$ is constructed. The measurement vector z is identical to the kinematic parameters x . There also are measurements of the height h provided by the edge-based adjustment. These measurements are processed independent of the planar motion model.

Error models are maintained over time by the Kalman Filter. As more images become available, the filter updates the vehicle model and associated error models. Errors are modeled by the Kalman Filter through the covariance matrices P_i , Q_i , and R_i . P_i is a measure of the error in the estimate \hat{x}_i and is maintained over time by the Kalman Filter. The filter is initialized by calculating \hat{x}_1 using the first two image frames. P_1 is set equal to the noise in the initial measurement (R_1), since the initial estimate \hat{x}_1 is determined entirely from the image information. Q_i is a model of the noise in the system state equation. This covariance matrix is constant over time ($Q_i = Q_0$) and was determined empirically. The associated standard deviations are $\sigma_x = [0.03m, 0.03m, 0.09m/s, 3.16^\circ, 1.72^\circ/s]$. The velocities were given small values in order to obtain a smooth vehicle trajectory. R_i is a model of the noise in the measurement equation. This matrix is determined based upon the quality of the vehicle model fit to the image information for the current measurement z_i . The quality of this fit is measured by extracting edges in the vicinity of the vehicle model and performing an edge-based adjustment. The covariance matrix R_i is defined as

$$R_i = (max - e_i) R_{max}$$

where e_i is the edge-based error measure, max is the maximum possible error measure, and R_{max} is the diagonal covariance matrix associated with the best possible measurement. The standard deviations associated with R_{max} were defined as $\sqrt{10}\sigma_x$, with the exception of the velocities v and w which were defined as $10\sigma_x$.

P_i is used within the Kalman Filter for determining the amount of weight to be placed upon new estimates of the system state. In addition, P_i is used to determine the maximum allowable edge-based position adjustment in 3-D space of the vehicle model. In Section 6.4.2 an edge-based adjustment algorithm was presented. This adjustment was based purely upon the image information. As the estimate of the vehicle position and motion is refined, the magnitude of this

adjustment can be restricted. The adjustment is restricted using the standard deviation values associated with the position parameters p_x , p_y , and h . The maximum allowable adjustment in 3-D space is defined as $3 \cdot \sqrt{\sigma_{p_x}^2 + \sigma_{p_y}^2 + \sigma_h^2}$. The adjustment perpendicular to the plane of motion is further restricted by defining a maximum allowable perpendicular adjustment of $2 \cdot \sigma_h$.

6.4.5 Results

The model-based vehicle tracking system was tested on a forty-five image sequence. This sequence was shot from a hand-held camera and contains a turning vehicle. The sequence was chosen because rotational motion is the most difficult case for the local translation tracker. The focal length of this camera was unknown. The focal length is another parameter that the user can manipulate during the tracking process. The results shown in this chapter used a focal length of 950 pixels (field of view of 30.2 degrees).

Initially, a vehicle model and a gravity model were instantiated by a user. The position of the gravity model is shown in Figures 6.18 and 6.19.

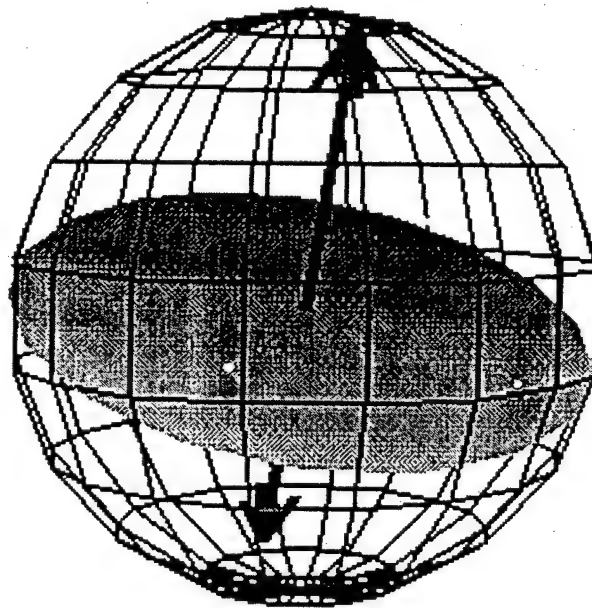


Figure 6.18. User-instantiated gravity model

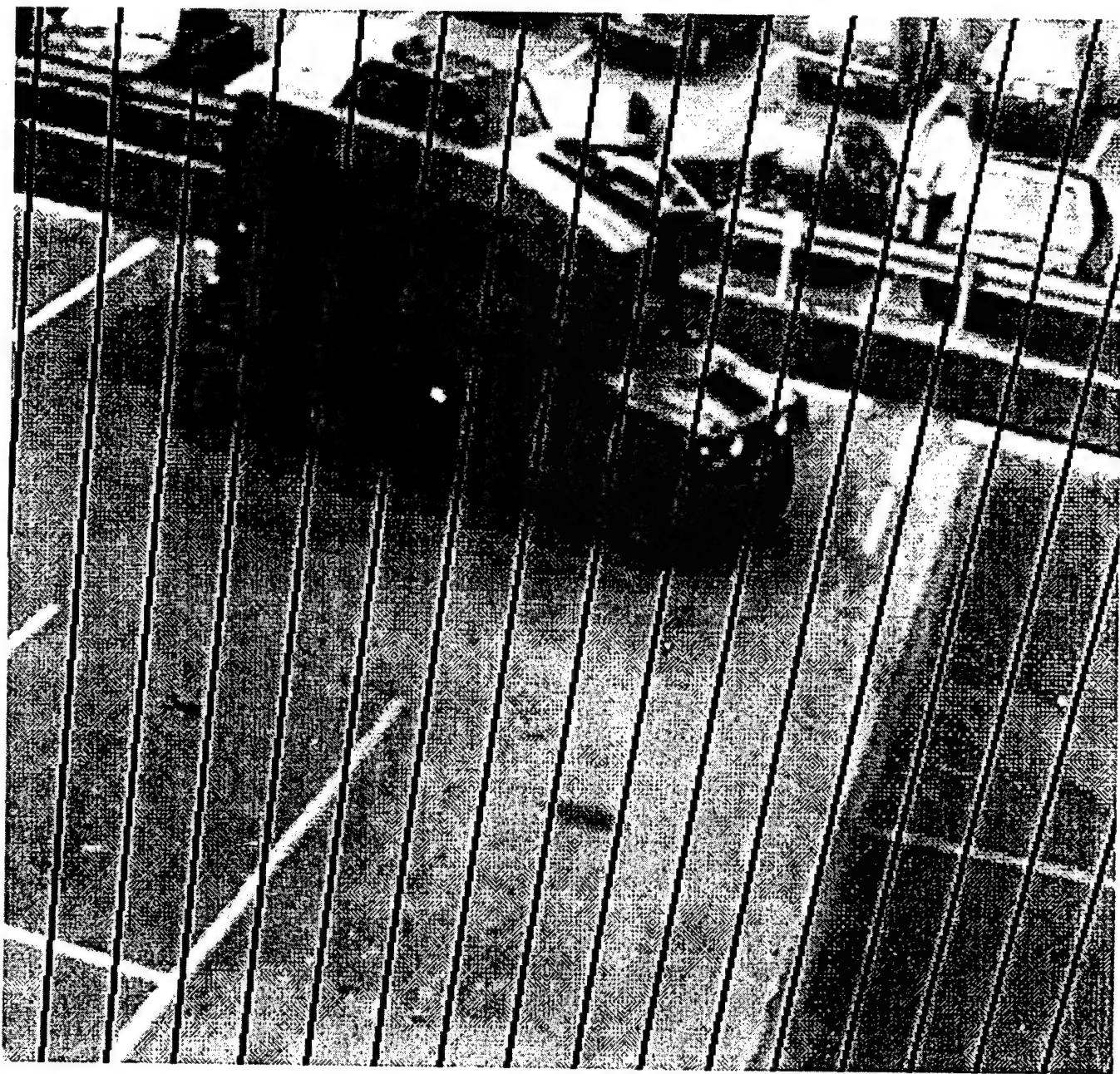


Figure 6.19. User-instantiated gravity model overlayed on image frame 1.

The initial vehicle position is shown in Figure 6.20.



Figure 6.20. User-instantiated vehicle model.

The edge-based position adjustment discussed in Section 6.4.2 was then applied to the image at the user-instantiated vehicle model position. The resulting vehicle model position is shown in Figure 6.21.

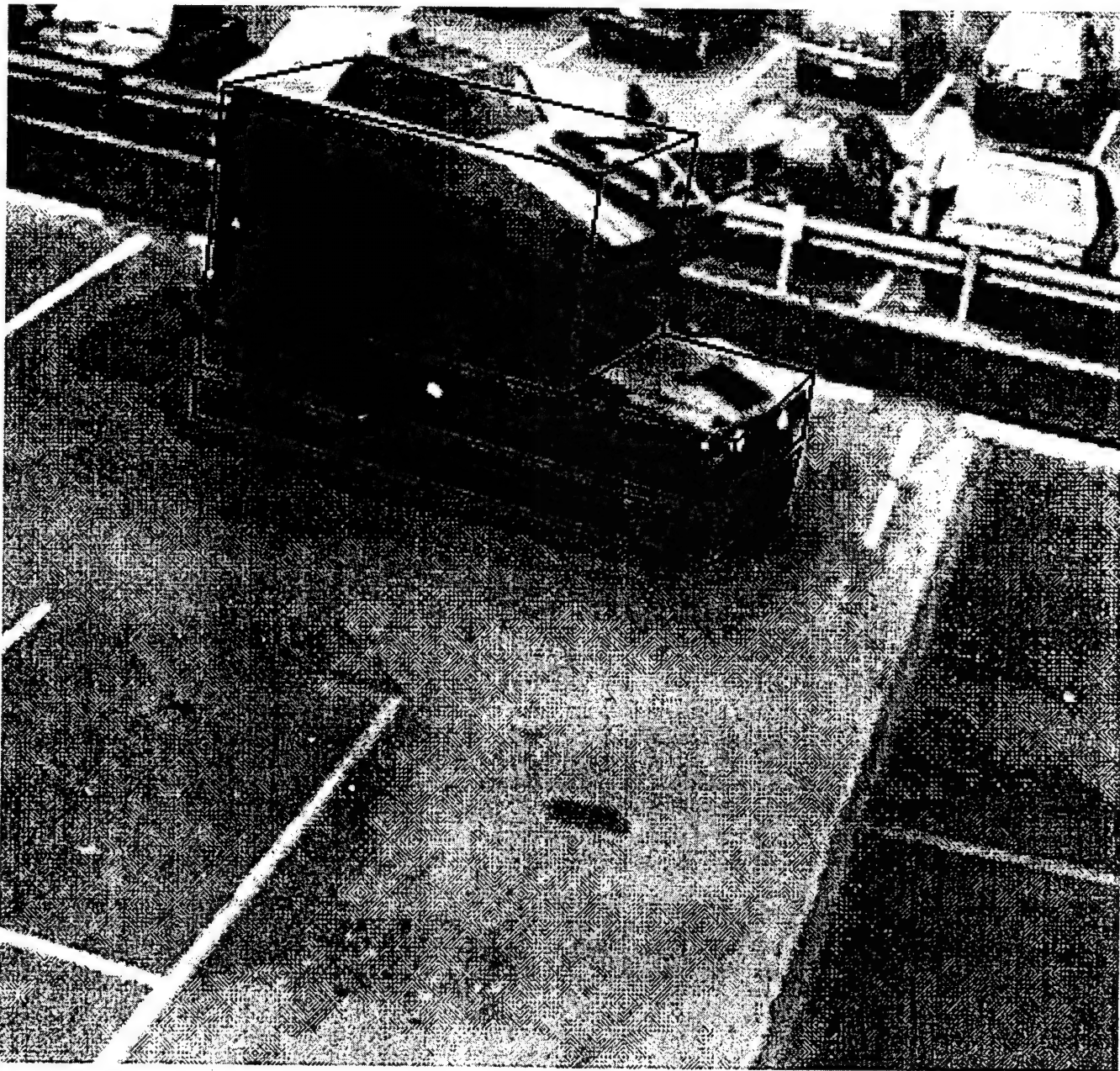


Figure 6.21. Adjusted vehicle model for image frame 1.

Using the vehicle model to control the extraction and group the results, features were extracted as discussed in Section 6.4.2. Four groups of features were formed representing the front, middle, and rear of the vehicle, as well as background features. The extracted and grouped features are shown in Figure 6.22. The background features were used to register the images by assuming that their motion was the result of translational motion parallel to the image plane.

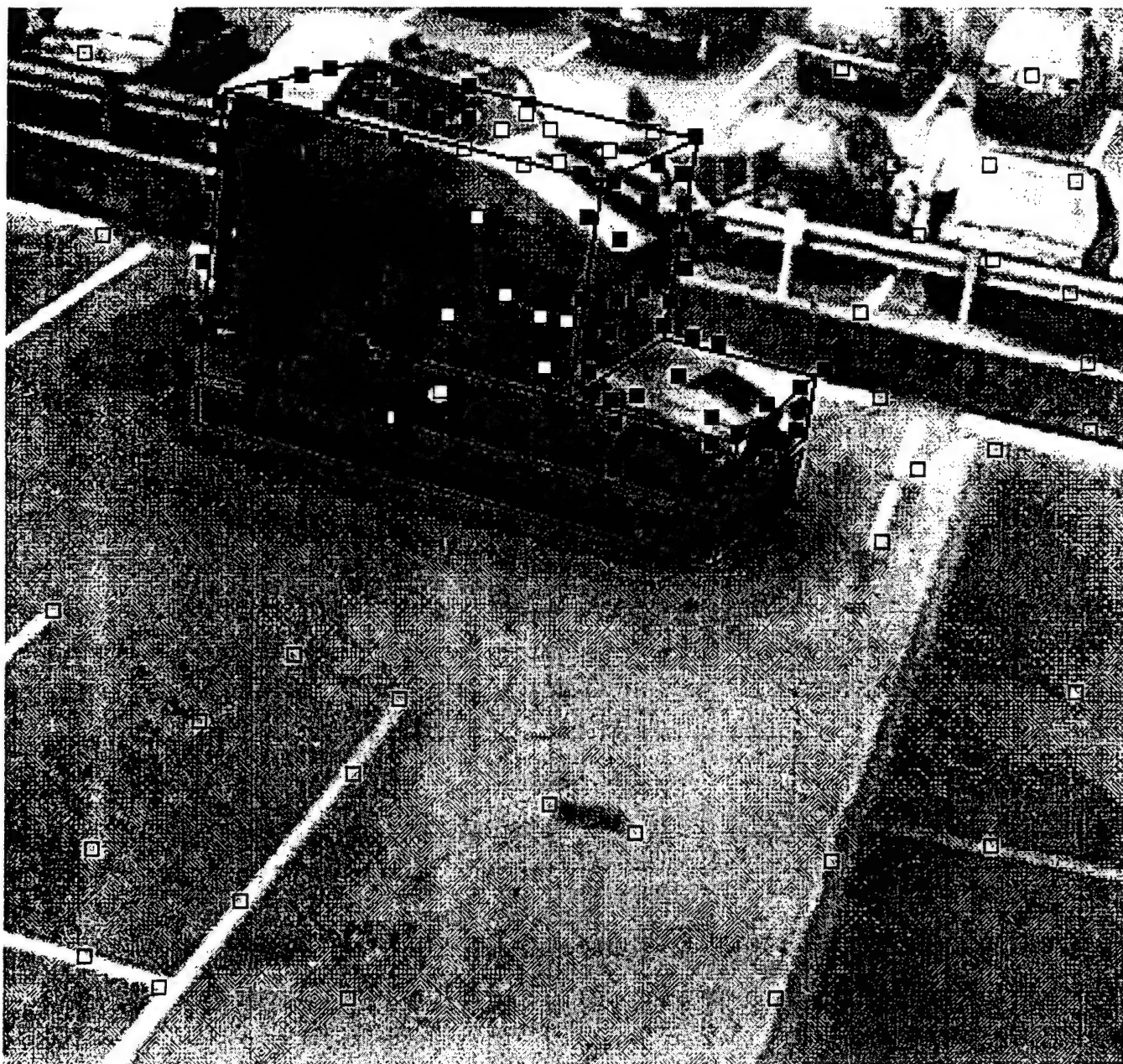


Figure 6.22. Extracted and grouped features for image frame 1.

Once the camera motion between frames 1 and 2 was calculated, a translational motion assumption was used to estimate the motion of the three groups of vehicle features. The search for an axis of translation was constrained by the direction of gravity and the current orientation of the vehicle. The plane perpendicular to the direction of gravity was searched in an area near the current vehicle orientation for the three local translation vectors. The results of this first search are shown in Figure 6.23. The top sphere contains the local translation vector resulting from the features on the front of the vehicle, the middle sphere corresponds to the middle of the vehicle, and the bottom sphere corresponds to the rear of the vehicle.

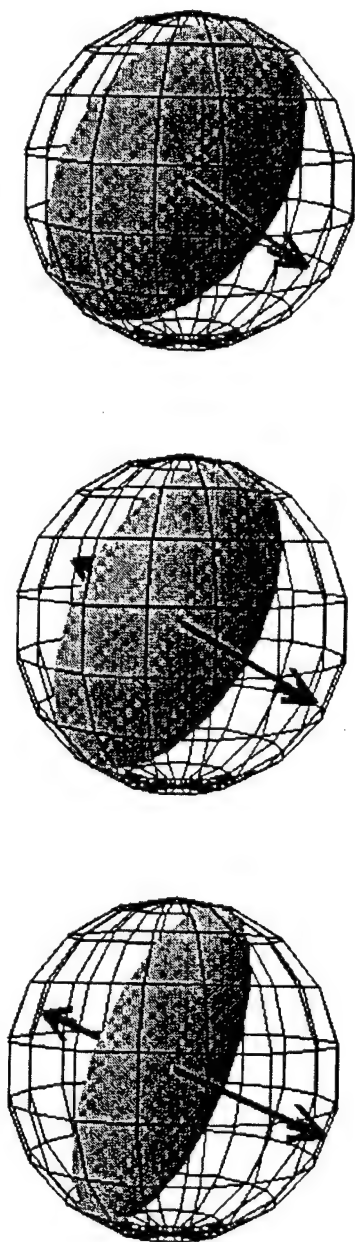


Figure 6.23. Local translation vectors for the three feature groups.

The three local translation vectors were used to calculate a new vehicle model position and orientation. The linear and angular velocities were also derived, and the Kalman Filter model was initialized using the results. This process was repeated for image frames 2 and 3, except that the local translation vectors were used to calculate only the orientation and velocity parameters. The vehicle position at time 3 was calculated using the edge-based adjustment algorithm. The location of the vehicle at time 3 was predicted using the kinematic estimate provided by the Kalman Filter at time 2. This predicted position was used to extract edge information and perform the edge-based adjustment in image frame 3. The resulting position estimate was passed to the Kalman Filter along with the orientation and velocities derived from the local translation vectors. This process was repeated for the remaining image frames in the sequence. Figures 6.24 through 6.27 show the position of the vehicle model at image frames 10, 20, 30, and 40, respectively.

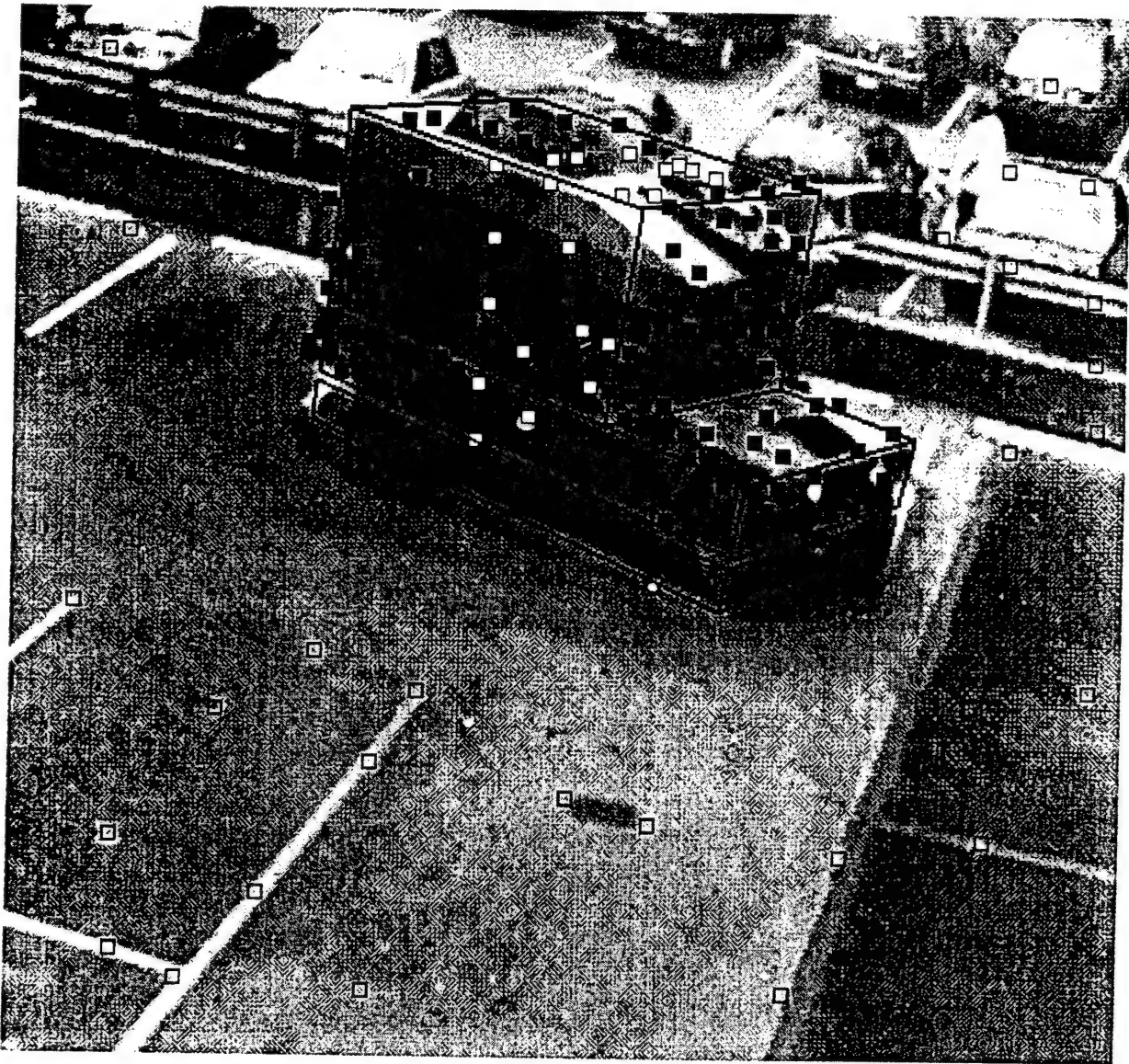


Figure 6.24. Vehicle model and features for image frame 10:

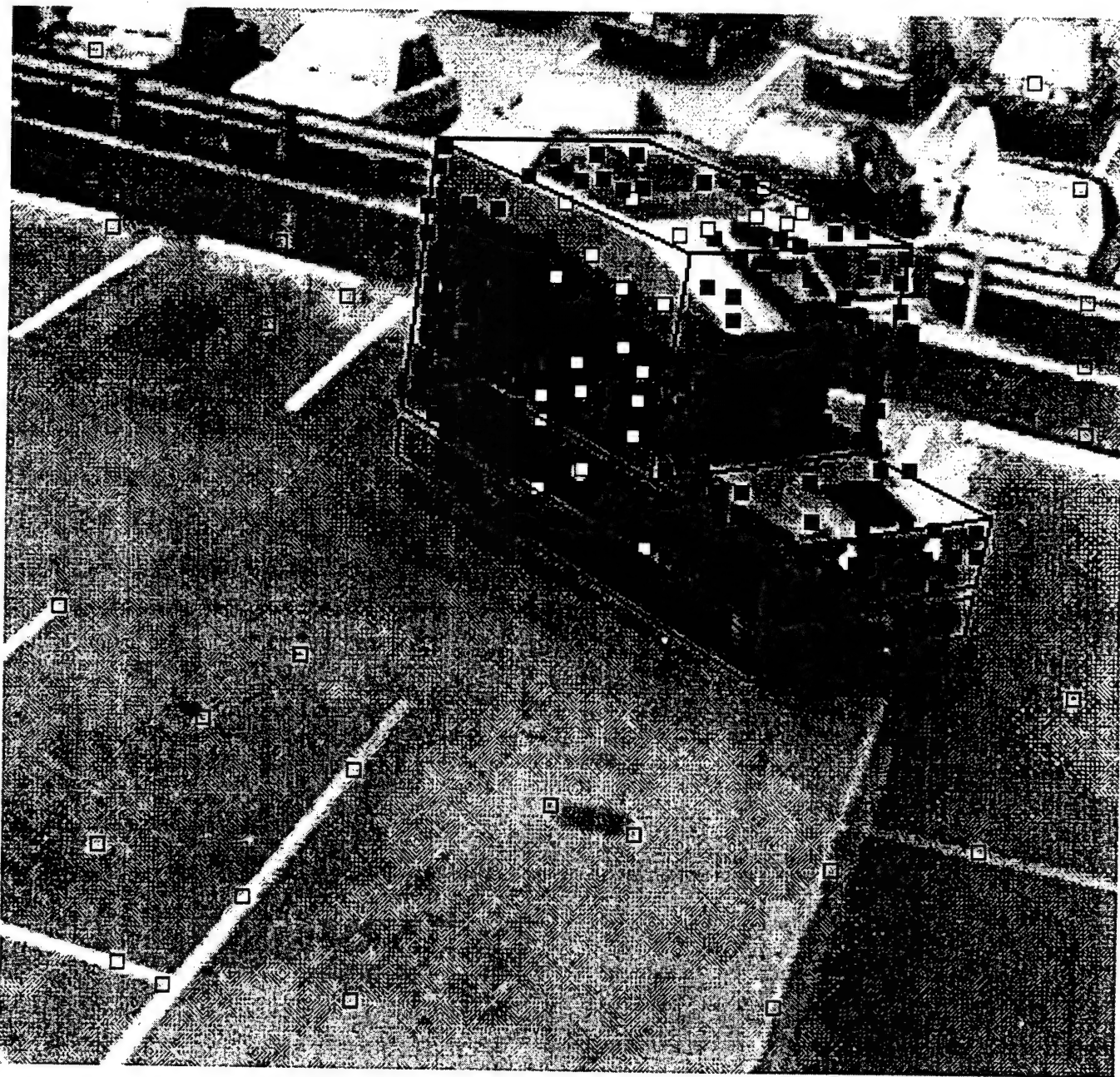


Figure 6.25. Vehicle model and features for image frame 20.

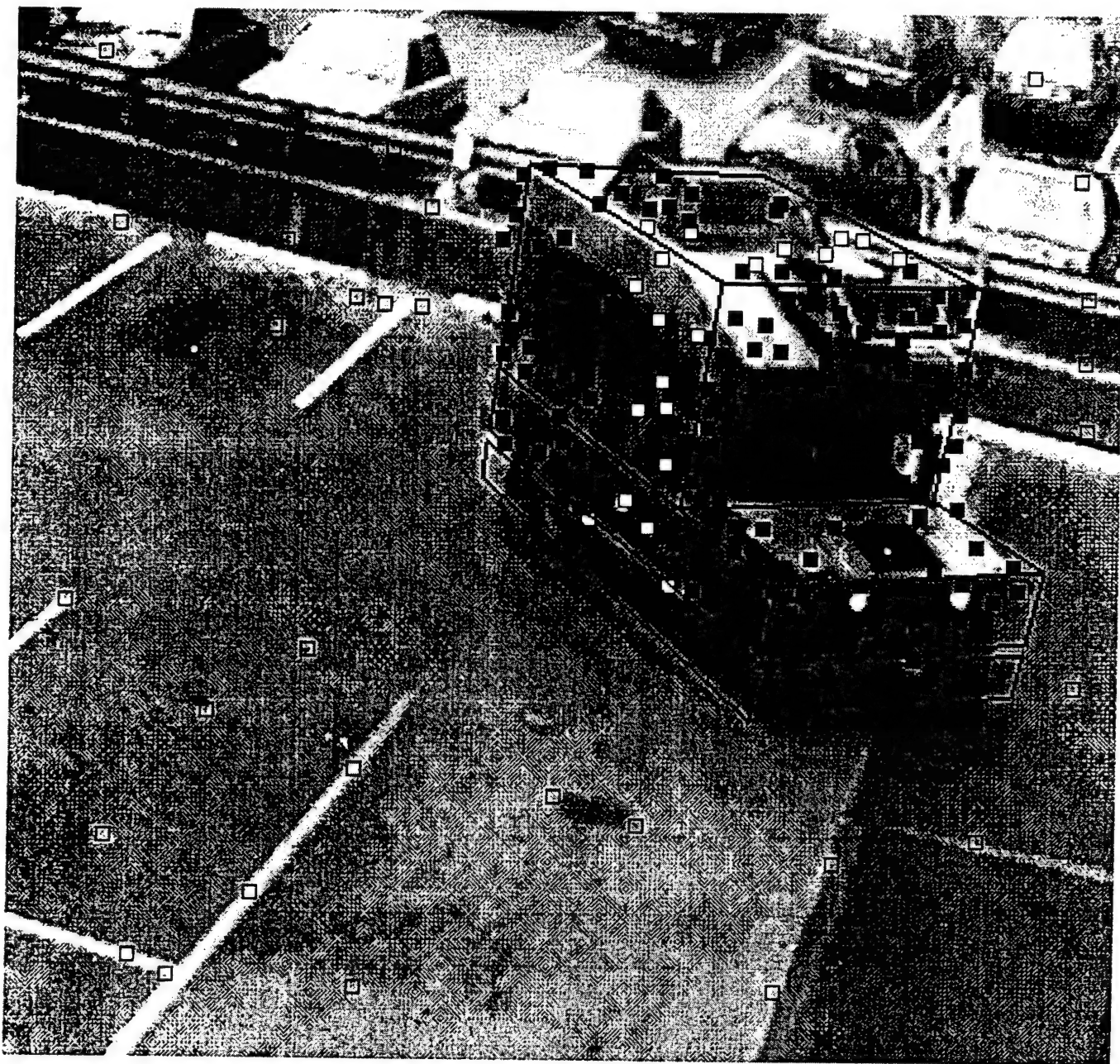


Figure 6.26. Vehicle model and features for image frame 30.

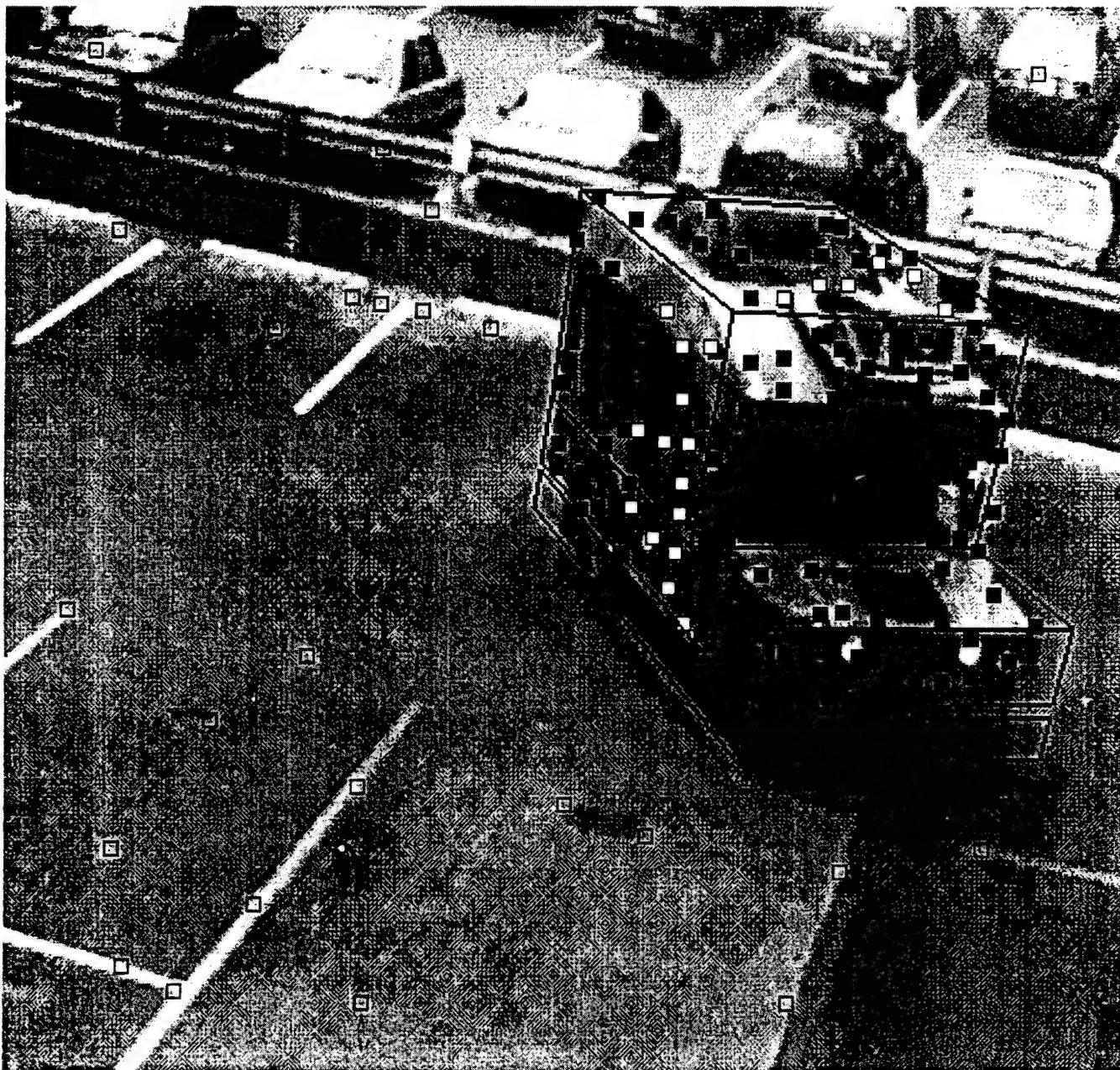


Figure 6.27. Vehicle model and features for image frame 40.

The actual vehicle length was unknown, so a value of 4 meters was chosen in order to calculate motion statistics. The remaining system parameters then were scaled relative to the truck length. Figure 6.28 shows the forty-five vehicle positions projected onto the plane perpendicular to the direction of gravity.

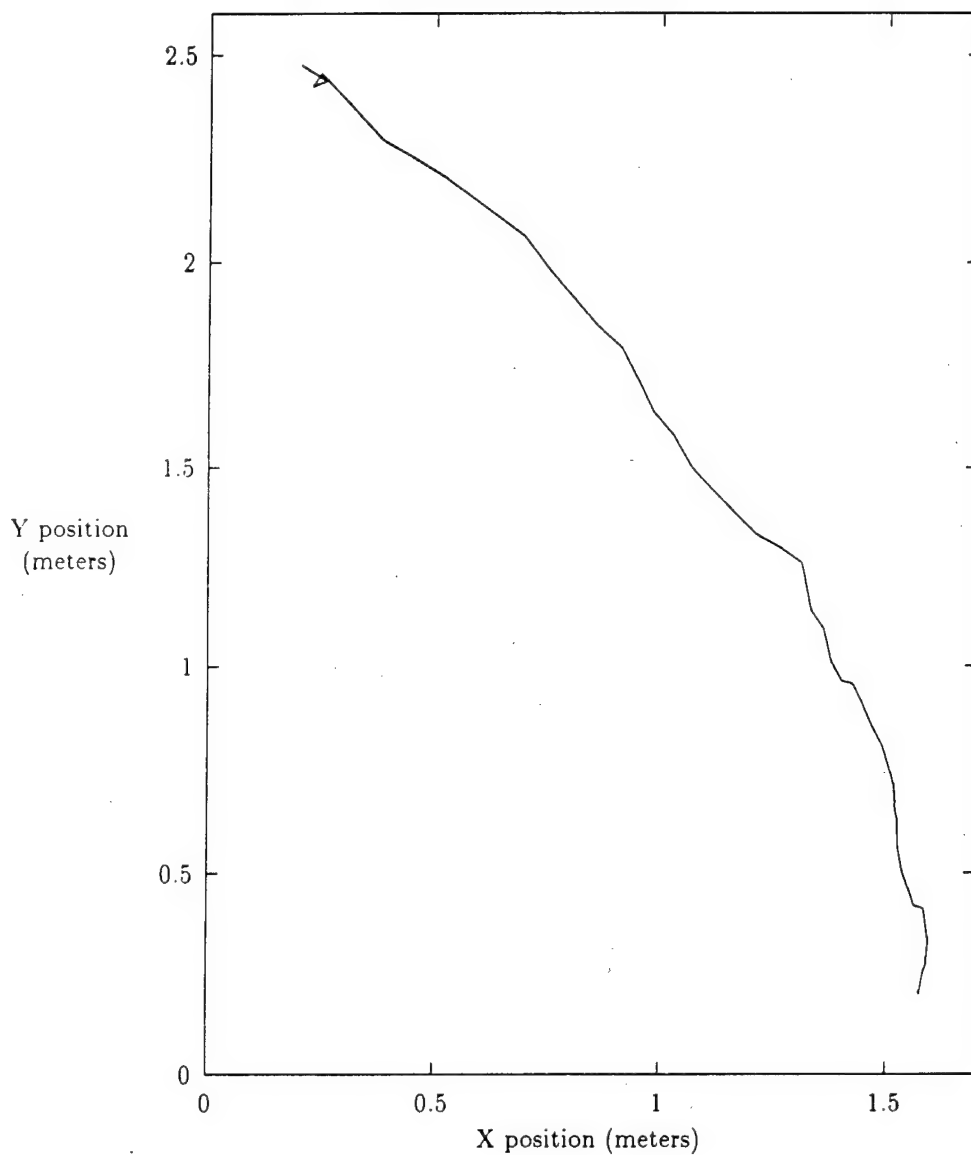


Figure 6.28. Vehicle trajectory for the 45 frame sequence.

The vehicle starts in the upper left hand corner of the plot and moves to the lower right corner. The height of the vehicle above the gravity plane is shown in Figure 6.29.

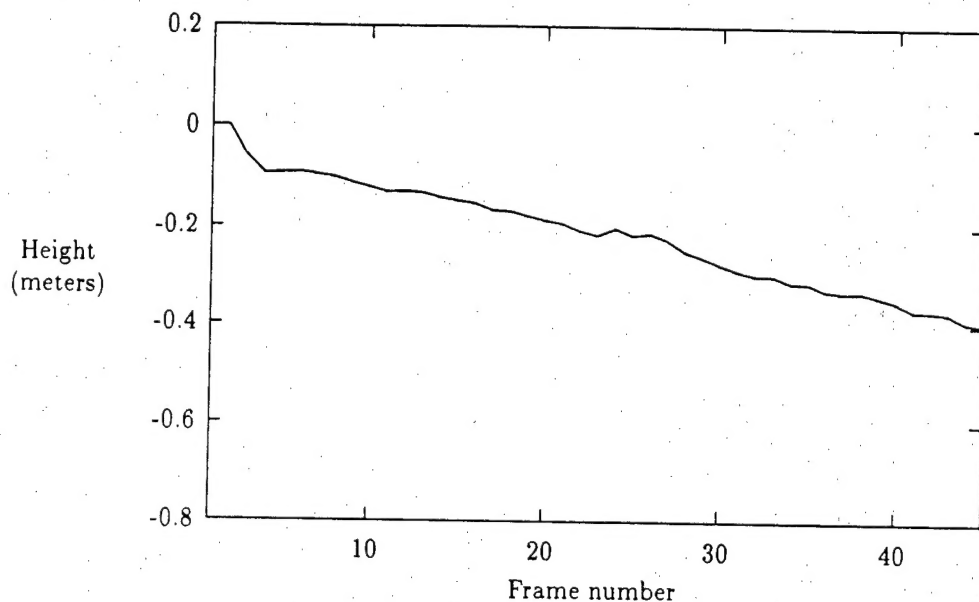


Figure 6.29. Height of the vehicle above the gravity plane.

This figure shows that the vehicle is moving slowly away from the plane of gravity because of errors in the focal length, vehicle model, and the user-instantiated gravity plane. The height information shown in Figure 6.29 is presented to the user so the appropriate action can be taken. The user can then adjust any number of these parameters in an attempt to correct for the height error. The vehicle orientation ϕ is shown in Figure 6.30.

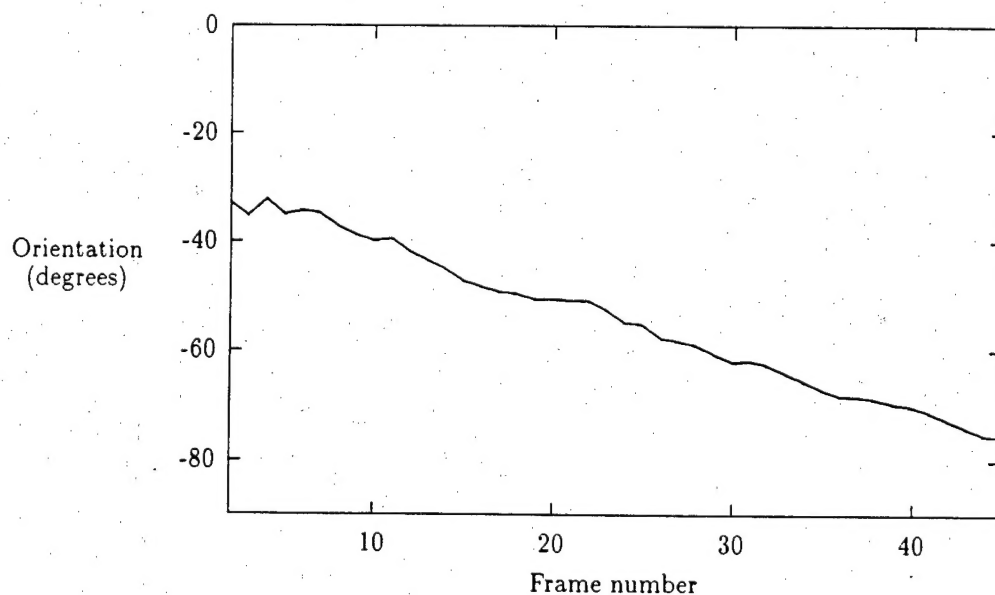


Figure 6.30. Vehicle orientation.

The orientation is measured with respect to the axis obtained by intersecting the image plane with the gravity plane. This axis corresponds to the x-axis shown in Figure 6.28. Finally, the linear and angular velocities of the vehicle for the forty-five frame sequence are shown in Figures 6.31 and 6.32.

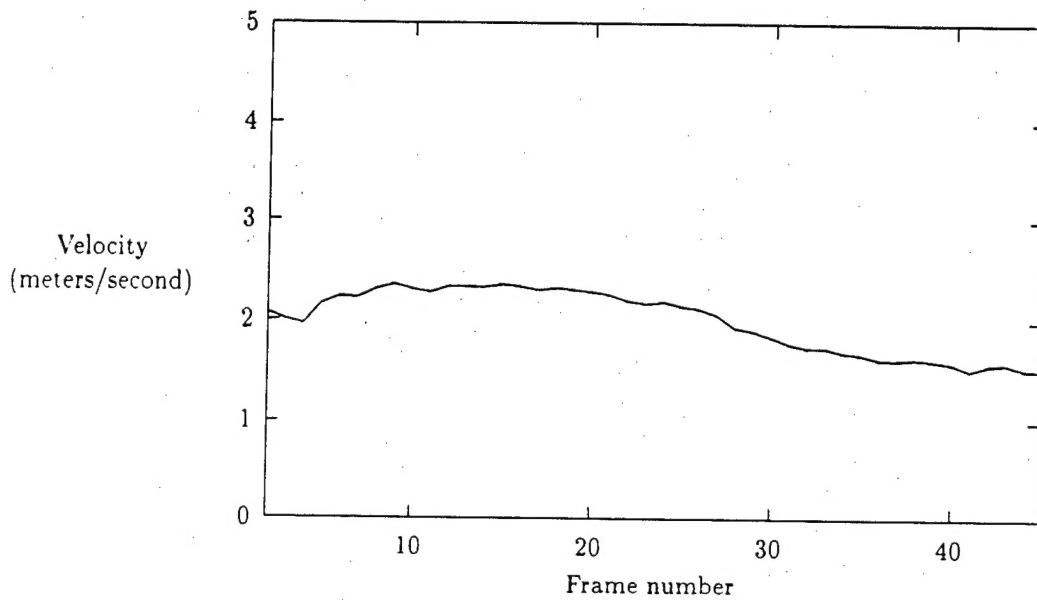


Figure 6.31. Linear velocity of the vehicle.

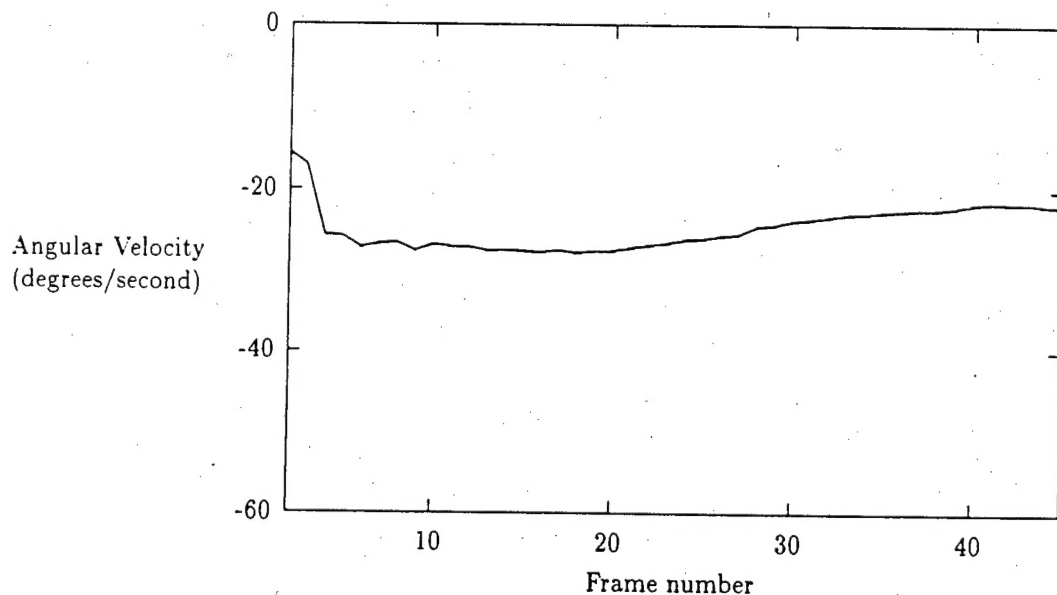


Figure 6.32. Angular velocity of the vehicle.

The results shown here demonstrate a robust model-based vehicle tracker, and show the feasibility of this technique for constructing simple yet robust solutions to difficult image understanding problems.

Bibliography

- [1] Ballard, Dana H. and Christopher M. Brown. *Computer Vision*. Prentice Hall, Inc., Englewood Cliffs NJ, 1982.
- [2] Borning, A.. The programming language aspects of thinglab, a constraint oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, 3(4):353ff, 1981.
- [3] Boulton, Terrance E. and Lisa Gottesfeld Brown. Motion segmentation using singular value decomposition. In *Proceedings of the DARPA Image Understanding Workshop*, pages 495-506, San Diego CA, January 1992.
- [4] Burger, W. and Bir Bhanu. On computing a 'fuzzy' focus of expansion for autonomous navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 563-568, 1989.
- [5] Jain, R. Direct computation of the focus of expansion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:58-64, 1983.
- [6] Kandel, Eric R., James H. Schwartz, and Thomas M. Jessell. *Principles of Neural Science*. Elsevier Science Publishing Company, New York NY, 1991.
- [7] Kuipers, B. J. and Y. T. Byun. A qualitative approach to robot exploration and map-learning. In *Proceedings of the Workshop on Spatial Reasoning and Multi-Sensor Fusion*, Los Altos CA, 1987.
- [8] Lawton, Daryl T. Constraint-based inference from image motion. In *Proceedings of AAAI-80*, 1980. Stanford CA.
- [9] Lawton, Daryl T. Processing translational motion sequences. *Computer Vision, Graphics, and Image Processing*, 22:116-144, 1983.
- [10] Lawton, Daryl T. and Warren F. Gardner. Translational decomposition of flow fields. In *Proceedings of the DARPA Image Understanding Workshop*, pages 697-705, Washington DC, April 1993.
- [11] Lawton, Daryl T., Warren F. Gardner, and Junhoy Kim. An interactive model based vision system for vehicle tracking. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 403-409, Atlanta GA, May 1993.
- [12] Lawton, Daryl T. and Todd S. Levitt. Knowledge based vision for terrestrial robots. In *Proceedings of the DARPA Image Understanding Workshop*, Palo Alto CA, May 1989.
- [13] Lawton, Daryl T., Todd S. Levitt, C. C. McConnell, P. Nelson, and J. Glicksman. Terrain models for an autonomous land vehicle. In *IEEE International Conference on Robotics and Automation*, San Francisco CA, April 1986.

- [14] Leler, W. *Constraint Programming Languages: Their Specification and Generation*. Addison-Wesley, Reading MA, 1988.
- [15] Levitt, Todd S. and Daryl T. Lawton. Qualitative navigation for mobile robots. *Journal of Artificial Intelligence*, 1990.
- [16] Mundy, J., P. Vrobel, and R. Joynton. Constraint-based modeling. In *Proceedings of the DARPA Image Understanding Workshop*, Stanford CA, May 1989.
- [17] Rock, Irvin. *Perception*. Scientific American Books, New York NY, 1984.
- [18] Ruoff, C., J. Bowyer, T. Brooks, J. Hanson, K. Holmes, and B. Wilcox. Autonomous ground vehicles: Control system technology development. Technical report, Jet Propulsion Laboratory, Pasadena CA, October 1984.
- [19] Steeb, R., S. Cammarata, S. Narain, J. Rothenberg, and W. Giuarla. Cooperative intelligence for remotely piloted vehicle fleet control, analysis, and simulation. Technical report, Rand Corporation, Santa Monica CA, 1986.
- [20] Tomasi, Carlo and Takeo Kanade. Shape and motion without depth. In *Proceedings of the DARPA Image Understanding Workshop*, pages 258-270, Pittsburgh PA, 1990.
- [21] Tomasi, Carlo and Takeo Kanade. The factorization method for the recovery of shape and motion from image streams. In *Proceedings of the DARPA Image Understanding Workshop*, pages 459-472, San Diego CA, January 1992.